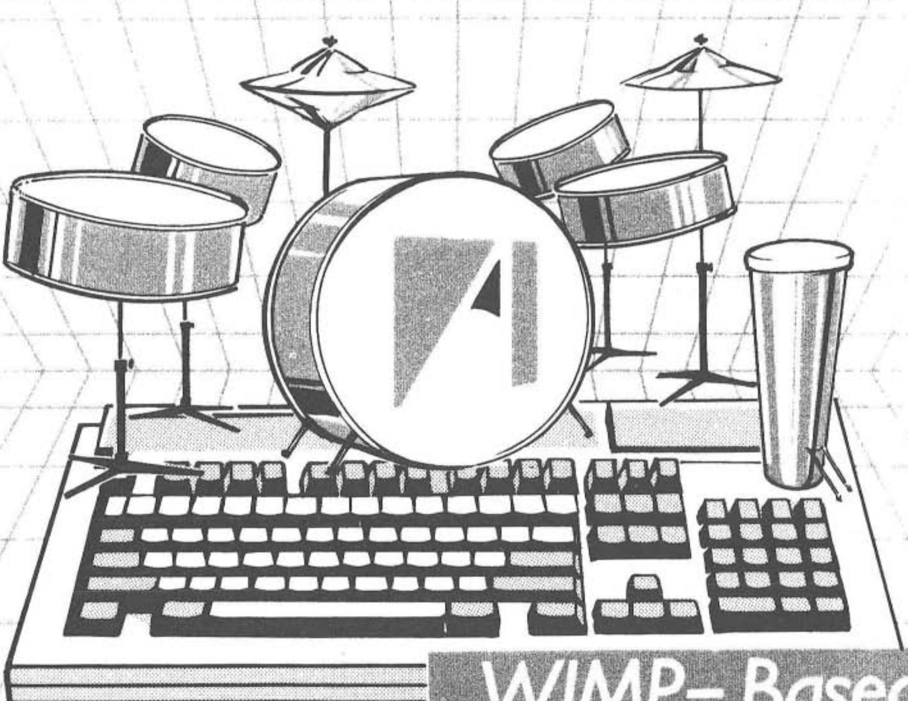
Volume 2
Issue 1

November
1988

Price £1.20

RISC USER



WIMP- Based Sound Sequencer

THE MAGAZINE AND SUPPORT GROUP
EXCLUSIVELY FOR USERS OF THE ARCHIMEDES

olume 2 Issue 1

vember 1988

RISC USER

CONTENTS

FEATURES

UTILITIES AND APPLICATIONS

REVIEWS

ADVERTISERS

RISC User is published by BEEBUG Ltd.

Co-Editors: Mike Williams, Dr Lee Calcraft
Assistant Editor: Kristina Lucas
Technical Editor: David Spencer
Technical Assistant: Lance Allison
Subscriptions: Mandy Mileham

BEEBUG, Dolphin Place, Holywell Hill, St.Albans,
Herts AL1 1EX. Tel. St.Albans (0727) 40303

Printed by Arion House

The Archimedes Magazine and Support Group.

EDITORIAL

This is the first issue of Volume 2 of RISC User, and you will find included a complete index to the first ten issues of Volume 1. Since we started, the Archimedes has established itself as a highly desirable and innovative machine, and Acorn certainly deserve praise for what it has achieved.

We have said on several occasions that it would be the availability of good software which would ultimately determine the Archimedes' success or failure. Much has already been achieved in this field, but only now are we beginning to see applications which really do show what the system can do. Acorn leads the way with RISC OS, the new multi-tasking version of Arthur due for release next April. Praise must also go to Clares Micro Supplies, where Dave Clare has done more than many to exploit the potential of the Archimedes. If you have not yet had a chance to see the current culmination of Clares' efforts, then ProArtisan is a treat which is still in store for you. Certainly, exciting times seem to lie ahead for the Archimedes.

We always welcome your own views and comments on what we do, and we would particularly like to encourage more readers to consider contributing to the magazine. Articles, programs, hints: all are welcome. In addition, we need some good writers to undertake some of the reviews we have planned for the future. If you think you can help, why not contact us?

The BBC Micro User Show, with particular emphasis on the Archimedes, takes place at the New Horticultural Hall, Westminster from 11th-13th November. We hope to meet as many members as possible on the BEEBUG/RISC User stand.

*This month's telesoftware password is **ferryboat**.*
(see BEEBUG pages on Micronet)

FASTER LANGUAGES

Hot on the tail of ABC (see last month's RISC User) comes another Basic V compiler, this time from Silicon Vision. *RiscBASIC*, as the new compiler is called, is claimed to cope with almost all Basic V programs. The only statement not supported is EVAL, and there is no limit on variable and array sizes (except that imposed by available memory). Full runtime error handling is included, as is the ability to enter the Basic editor whenever a compilation error occurs.

Also new from Silicon Vision is *RiscFORTH*, a complete implementation of the FORTH-83 language. This was previously released by Blue-Grey Software, but Silicon Vision has bought the rights and is now the sole publisher. A major feature of *RiscFORTH* is the ability to write programs that can run concurrently with each other. Both *RiscBASIC* and *RiscFORTH* cost £99.95 inc. VAT each, and are available from Silicon Vision Limited, who are located at Signal House, Lyon Road, Harrow, Middlesex HA1 2AG, tel. 01-422 2274 or 01-861 2173.

ACORN BACK IN THE BLACK

Acorn made a profit of £711,000 in the first half of 1988, compared with a loss of nearly £1 million in the same period last year. This has meant that Acorn has been able to cut its bank borrowing by £1 million in the last year. While the total sales for this period showed only a slight increase to £20.5 million, Acorn's chairman Elserino Piol is confident that this is only the start. Since these figures were released, income from Archimedes sales has exceeded that of the Master series for the first time. Hopefully, this will ensure a bright future for Acorn, and provide security to its customers.

GRAPHS GALORE

Mouse Plotter is a new graph plotting package from the Shell Centre for Mathematical Education at Nottingham University. Data can be read into *Mouse* Plotter *from a number of different file* formats, including plain text. It is also possible to read in raw numbers, and the program automatically chooses the correct data type. Expressions to be plotted can be in the usual form of y=f(x), for example y=SIN(x), or a more complex form such as the equation $y^2=a^2-x^2$. Graphs can be scaled automatically, and areas of interest can be blown up by dragging a box with the mouse. *Mouse Plotter* costs £15 (inc. VAT) and is available from ITMA, Shell Centre for Mathematical Education, University of Nottingham, Nottingham NG7 2RD.

ARCHIMEDES ART

Beard Technology has released a mode 15 drawing package called *Leonardo 256* to complement the original *Leonardo* which runs in mode 12. Both packages include commands to draw various shapes, as well as drawing freehand, and there are also facilities such as a flood fill. A zoom facility allows all editing to be performed at any level of magnification, and if you make a mistake there is an undo option which can itself be undone. A special data compression technique can be used to save pictures, thereby allowing many more to be fitted on each disc. *Leonardo 256* costs £19.50 (inc. VAT), while the original *Leonardo* is available for £17.50 (inc. VAT). Both of these packages can be obtained from Beard Technology who are situated at 111 Evering Road, London N16 7SL, or tel. 01-806 4460.

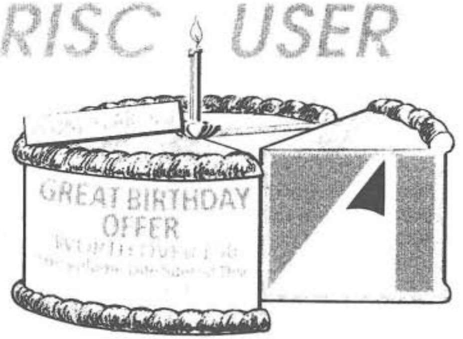
THE FUTURE OF ECONET

At the recent Econet '88 conference in Birmingham, Acorn emphasised its commitment to the Econet network system, especially with the Archimedes. The new network drivers in RISC OS cure many of the problems that Archimedes network users had complained about, and Acorn is set to release a new Filestore system to act as a network file-server. The new Filestore is claimed to be much faster than the current model, and can support up to 240Mbytes of disc storage via a series of 40 and 60Mbyte hard discs. Acorn also stated that it is working on connecting the Archimedes to faster networks such as Ethernet, which will allow the full speed of the ARM processor to be utilised.

PHONE IFEL

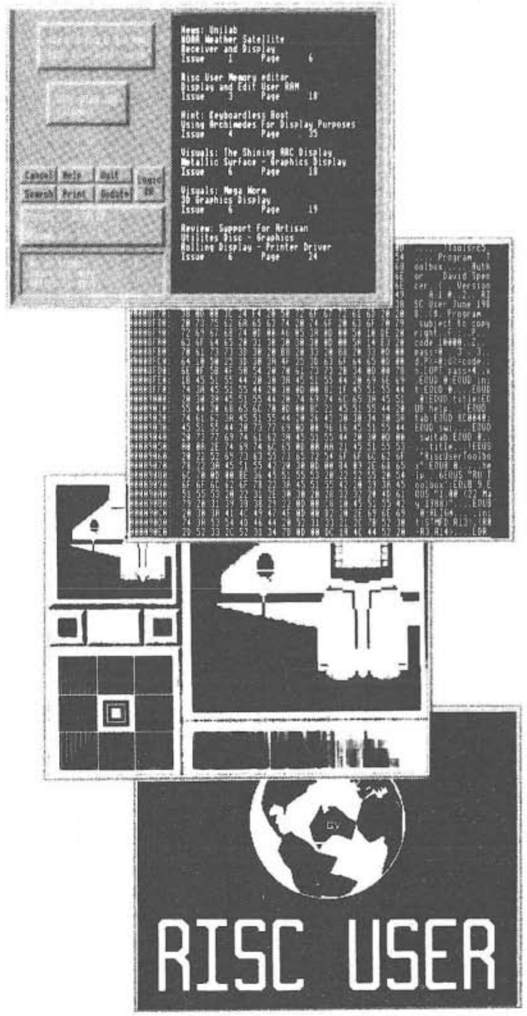
Several RISC User members have experienced problems contacting IFEL, manufacturers of the four-slot backplane featured in last month's news. IFEL has in fact changed its phone number. The new number is (0752) 847286, and not the one given in last month's news, or in IFEL's advertisement.

RISC USER



The special disc to celebrate our 1st birthday will be available with the November issue.

SPECIAL DISC CONTAINS ALL THIS FOR JUST 4.95



```
News: Unilab
NOAA Weather Satellite
Receiver and Display
Issue     1      Page      6

Risc User Memory editor
Display and Edit User RAM
Issue     3      Page      18

Hint: Keyboardless Boot
Using Archimedes For Display Purposes
Issue     4      Page      35

Visuals: The Shining ARC Display
Metallic Surface - Graphics Display
Issue     6      Page      18

Visuals: Mega Worm
3D Graphics Display
Issue     6      Page      19

Review: Support For Artisan
Utilites Disc - Graphics
Rolling Display - Printer Driver
Issue     6      Page      24
```

1 ARCSCAN

A fast on-screen bibliography with powerful search facilities for all the RISC User and BEEBUG magazines. Normal price £12.

2 PIXEL EDITOR

This powerful drawing tool is a full screen full-feature pixel editor for creating and editing screens and sprites.

3 TOOLBOX

This incredibly useful utility features a memory editor, memory search and replace, disc editor and disassembler. TOOLBOX contains many of the features found in packages costing over £35.

4 WORLD IN MOTION

A stunning animation with an oddly reminiscent feel to it.

5 DISC MENU MODULE

Use the mouse to control your disc files with this extremely useful relocatable module.

6 PRINTER BUFFER

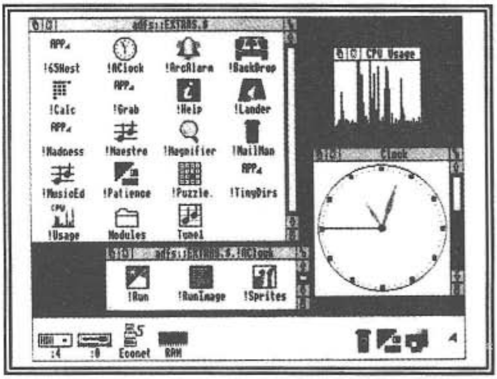
This printer buffer frees your computer during long printouts and is configurable from a few bytes to 4 Mbytes. Similar to packages currently selling at £19.

Altogether the items on the disc would be worth over £50 if bought separately. See your subscription reminder or next month's magazine for full details.

RISC OS REVEALED

David Spencer gives a preview of Acorn's exciting new operating system for the Archimedes.

I am sure that most Archimedes owners are aware by now of RISC OS, the new operating system which Acorn unveiled at the Personal Computer Show in September. The purpose of this article is to give an insight into the new facilities offered by RISC OS.



MULTI-TASKING

One of the most talked about features of RISC OS is multi-tasking - the ability to execute two or more programs concurrently. The first thing to say is that RISC OS does not support genuine multi-tasking. Instead, the RISC OS approach to multi-tasking centres around the use of the Window manager (WIMP). The key to any WIMP based program is the use of the SWI call 'Wimp_PollWimp', which is provided by the WIMP module. Essentially, an application program continuously calls this routine which then returns a code to indicate what action should be taken next. This may be something like redrawing a certain window, or processing a keypress. Alternatively, if the WIMP doesn't need any tasks performed, the application can do its own processing, for example, updating a clock display.

In Arthur 1.20, whenever an application called the WIMP polling routine, it would return almost immediately with a request for whatever action should be performed. This is not true for RISC OS, which instead maintains a list of active tasks. Each time one of these tasks calls the polling routine, the WIMP returns not to the calling task, but to the next one in the list. This is repeated for all the tasks, until the original caller finally gets dealt with. The entire process is then repeated. This provides a way for all the applications to perform their functions without any knowledge of each other.

THE DESKTOP

The Desktop program in 1.2 was written in Basic and took up nearly 100K of the operating system ROMs. In contrast, the RISC OS Desktop is a mere 6K of ARM code, and rather than being an application in its own right, is more a 'spring-board' for starting genuine applications. When you enter the Desktop, it 'hunts out' any modules which contain applications and starts these up.

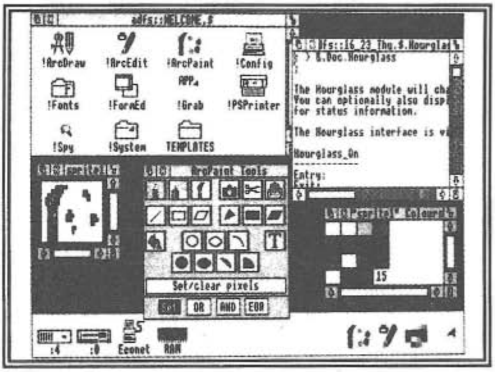
The ROM based applications stored as modules are the task switcher, the palette utility and the filer. The first of these is the all-important utility for switching between tasks. The task switcher is also responsible for allocating the correct amount of memory to each task. The amount of memory used for areas such as the screen and sprite workspace can be changed simply by dragging sliders on the screen.

The palette utility is similar to that from Arthur 1.20, but with the ability to operate in any mode. It is possible to switch from mode 12 to 15 within the Desktop, and hardly notice the change. Similarly, you can switch into mode 16 or 17 and get a much wider display. The Desktop will even work in mode 0 by using shading to represent colours.

The filer is probably the most important resident application. As on Arthur 1.20, directories can be displayed from any filing system, and sub-directories can be opened just by clicking on them. However, it is also possible to copy a file from one place to another just by dragging it between windows. Applications can be installed simply by double clicking on them, and they will then appear as an icon on the bottom icon bar. To open an installed application just click on its icon. A file can be loaded into an application simply by dragging it from one of the filer's windows onto either the application's window or its icon. To save files, each application pops up a window containing a icon for the file to be saved. This can then be dragged to the filer window of your choice.

FILING SYSTEM CHANGES

RISC OS also includes a number of improvements as far as filing systems are concerned. The most noticeable change is the addition of a RAM filing system. This stores files in a RAM disc, the size of which is configurable by the user. The RAM filing system behaves just like the ADFS in use, and files can easily be copied between the RAM disc and a real disc (or a network).



Ifs::16_23_Thu.$.Hourgla
> $.Doc.Hourglass

The Hourglass module will ch
You can optionally also disp
for status information.

The Hourglass interface is v

Hourglass_On

Entry:
Exit:

The ADFS has also been extended to allow an extra format. This new 'E' format, as it is called, stores 800K per floppy, just as the current 'D' format does. However, the new format does not necessarily store files as one continuous block. Instead, a 'scatter map' technique is used to allocate disc storage. The directory entry for each file now contains a file number rather than the disc address of the file. This file number is used to locate a map on the disc which shows where all the parts making up a complete file are stored. Using this technique means that you no longer have to use *COMPACT, because a file can be split between areas of free space. Another advantage is that any defective sectors on a disc can be mapped out. Therefore, a single damaged sector no longer renders the entire disc unusable.

A further improvement to the ADFS is the ability to configure the amount of RAM used to buffer directories, and also the amount of RAM to be used as a file cache. By changing the file cache size it is possible to greatly reduce the number of disc accesses when performing certain random access operations on files.

For Econet users there have been a number of improvements to the network software. In particular, the Econet driver in RISC OS allows the transfer of files to be done in sections. This avoids the problem of one computer hogging the network while transferring a large file. Operations such as *NOTIFY are also implemented in the new software.

MISCELLANEOUS FEATURES

As well as the major changes already listed there are a number of other features new to RISC OS:

A drawing module that includes Bezier curves. This makes it easy to implement a page description language such as PostScript, and is thus ideal for driving laser printers.

A number of new SWI calls, including a high speed heap-sort routine.

International keyboard drivers to change the keyboard layout for different countries.

An improved serial port driver with all the previous bugs fixed.

Faster interrupt handling, allowing the use of faster hard-discs.

A new 6502 emulator that mimics a model B rather than a second processor.

WELCOME DISCS

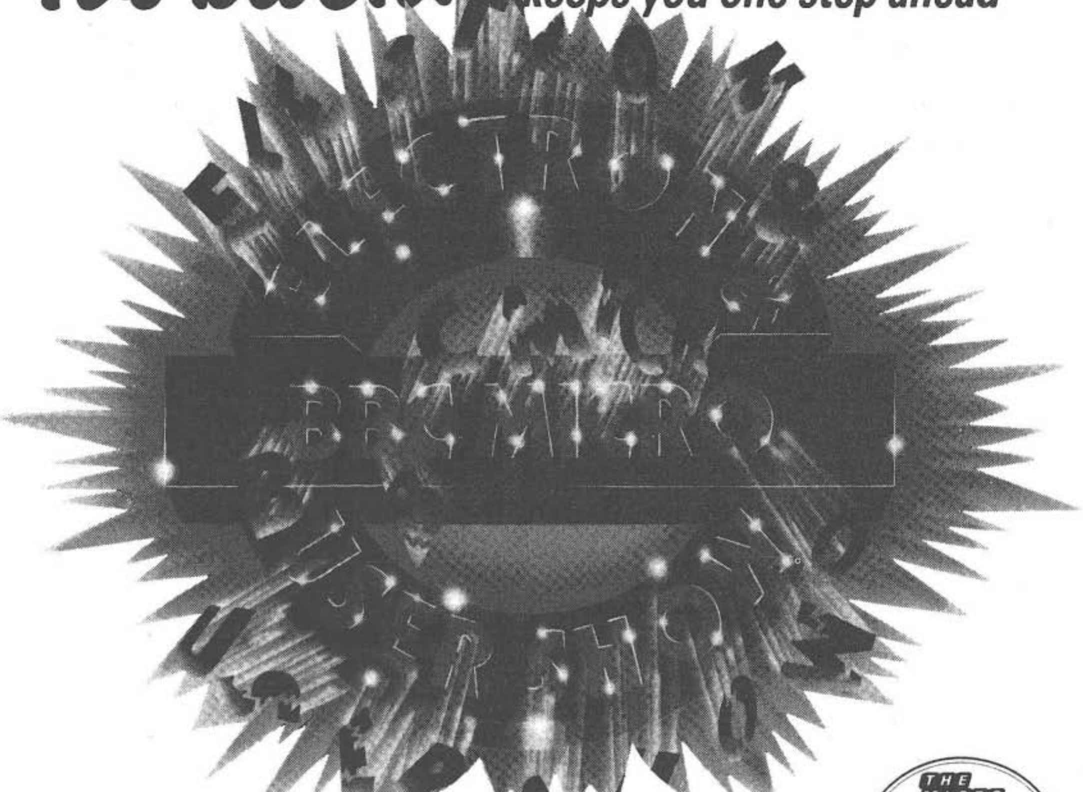
RISC OS will be supplied with two Welcome discs which will contain three major applications in addition to the usual examples and tutorials. These are ARCEdit, a full-feature text editor; ARCPaint, a sprite designer and painting package; and ARCDraw, an object based drawing package. All of these are very useful tools to complement RISC OS.

CONCLUSION

Overall, RISC OS provides a highly effective and user friendly working environment, not dissimilar to that of the much acclaimed Apple Macintosh. At the same time, Acorn has achieved almost total compatibility with the current Arthur 1.20.

NOTE:
RISC OS will not be available until April 1989
However we are accepting advance orders at
£33.35 inc. VAT (BEEBUG retail code 0919D).
Please add £3 for p&p

It's back! The show that ALWAYS keeps you one step ahead



ıe premier exhibition for users
all Acorn machines returns to
popular venue in the heart of
e capital.

Traditionally the liveliest event
the year on the Acorn
lendar, the pre-Christmas
ow is the one you just cannot
ford to miss.

s your value-for-money
ıssport to:

70 exhibitors displaying all
the latest developments
across the entire Acorn
range.

Archimedes World – which
provides a fascinating
glimpse into the current and
future roles for this
remarkable machine.

Technical advice from the
UK's leading experts on all
Acorn computers.

Hundreds of special offers for
the BBC Micro and Electron
waiting to be snapped up as
top-value Christmas presents.

this – and so much more –
the 20th record-breaking
ctron & BBC Micro User
ow

u can even save yourself £1
ore you get there by using
advanced ticket form.

New Horticultural Hall, Greycoat Street, London SW1

10am-6pm Friday, November 11
10am-6pm Saturday, November 12
10am-4pm Sunday, November 13

ELECTRON & BBC MICRO USER SHOW

Advance ticket order

SAVE £1

Please supply tickets for November show:

□ Adult tickets at £4 (save £1) .... £.......

□ Under-16s tickets at £2.50 (save £1) .... £.......

□ Cheque enclosed made payable to Database Publications Ltd Total £ ____

Admission at door:
£5 (adults)
£3.50 (under 16s)
Advance ticket orders must be received by November 2, 1988

□ Please debit my credit card account: □ Access □ Visa Expiry date: [/]

Name ....

Address ....

....

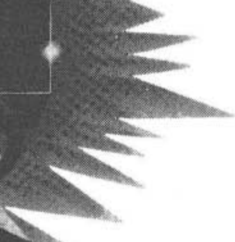
Signed ....

Post to: ***Database Exhibitions, Europa House, Adlington Park, Adlington, Macclesfield SK10 4NP.***

DATABASE EXHIBITIONS

PHONE ORDERS: Ring Show Hotline: 0625 879920
Prestel Orders: KEY *89 THEN 614568383
MicroLink/Telecom Gold Orders: 72:MAG001
Please quote credit card number and full address

A403



Take a stroll down Innovation Row – a brand new show feature area, specially constructed for the event.

See the grand finalists displaying their breakthroughs in public for the first time. And you can help pick the winners by casting a vote in both categories of the awards – BBC Micro and Archimedes.

How to get there

Underground: The nearest tube stations are VICTORIA (Victoria, District and Circle Lines), ST. JAMES'S PARK (District and Circle Lines) and PIMLICO (Victoria Line).

By British Rail: VICTORIA STATION. The halls are a 10-minute walk from the station.

By Bus: 11, 24, 29, 70, 76 and Red Arrow 507 to Victoria Street – alight Army and Navy Stores.

WIMP-Based Sound Sequencer

Peter Harris combines the power of the Archimedes WIMP manager with the audio capabilities of the sound system to produce a sophisticated rhythm system.

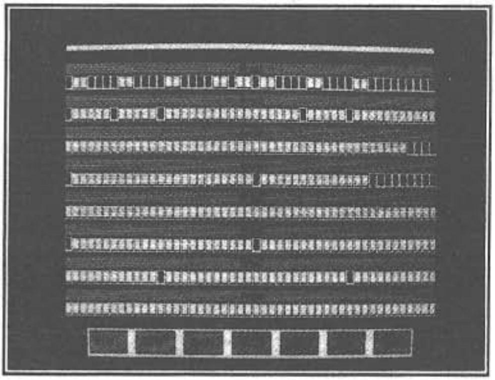
In RISC User Volume 1 Issue 7, a simple 'Beat Box' program was published for generating percussion rhythms. The program presented this month uses the same basic idea, but the screen display has been greatly enhanced using the Archimedes WIMP manager, while both default and user defined voices may be included and the resulting base rhythms saved to disc as required. In this way a library of rhythms may be built up and accessed as required. The whole system is very easy to use and gives excellent results.

In fact, the use of the WIMP manager is quite novel. The entire screen display is constructed from windows and icons, including all the text legends. You will find that there are no PRINT statements at all in the listing, and the program may well prove instructive to those interested in using the WIMP manager or the sound system themselves.

To start with, type in and save the program before running it. As listed, it uses the excellent RISC User percussion module (Volume 1 Issue 5 disc) to provide a comprehensive repertoire of suitable sounds (the module is referred to as 'PercusMod'). If this is not available, simply omit (or delete) lines 550 to 570 and 630 to 640, and the program will use the Archimedes' default voices. The screen display, as in the illustration, shows 64 beats for each of eight voices across the screen, and seven control icons at the foot of the display. The eight voices will be labelled with the voice names.

Using the mouse pointer and the *select* button, you can enter beats for any of the eight voices, or similarly clear any already set. Clicking on the *START* icon will set the rhythm going, while the *STOP* icon will terminate it. The other icons are equally self-explanatory, allowing the pace of the rhythm to be increased or slowed down, and the entire display can be reset (cleared) to its initial empty state.

Two further icons allow the current rhythm to be saved to disc (a window appears to prompt for the file name), or a previously saved rhythm can be reloaded. The magazine disc contains a number of rhythms saved in this way (some of them are 'empty' templates to allow your choice of rhythm to be entered), as well as the RISC User percussion module referred to earlier.

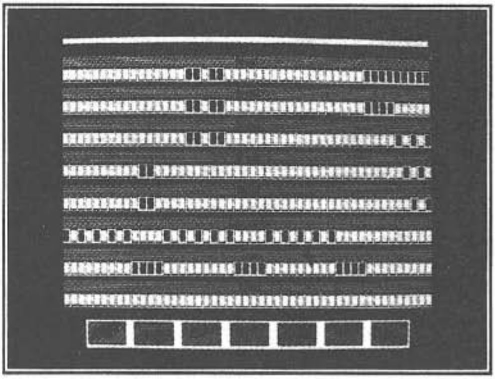


CHANGING PARAMETERS

Lines 90 and 100 contain various parameters which can be readily changed. The *speed* factor determines the rate at which the beats occur (the higher the value the slower the rhythm). The value assigned to *beattotal* controls the number of beats across the width of the screen display (beats per bar). The screen display automatically adjusts to whatever value is specified.

The line of DATA specifies which voices are to be used. By default the Archimedes provides eight voices, the maximum which may be active at any one time. The RISC User percussion module supplies 14 different voices, and any eight of these may be selected by specifying the corresponding voice number. You can see what voices (or instruments) are available by typing *VOICES. When you first switch on, this will show the default voices. If you run the program once with the percussion module, and then type *VOICES you will be able to see what is available there.

The power of the program lies in its flexibility. If you modify any of the parameters described, then the new values will be saved along with the beat information. On reloading, all this information is used to reconstitute the display according to the parameter settings in use when that rhythm was saved.



This program is a truly remarkable demonstration of what can be achieved by harnessing the sound system to the WIMP manager. For the future we hope to include further sets of voices on the monthly program disc.

```
 10 REM >SEQUENCE
 20 REM Program   Drum Sequencer
 30 REM Version   A1.7
 40 REM Author    Peter Harris
 50 REM RISC User November 1988
 60 REM Program   Subject to Copyright
 70 :
 80 REM Default values
 90 speed=20:beattotal=64
100 DATA 1,2,3,4,5,6,7,8
110 :
120 MODE12:*FX4,1
130 ON ERROR PROCerror
140 DIM window%(8),soundflag(8)
150 DIM voice$(8),channel(8)
160 DIM block 1000,fnbuffer 12
170 DIM errbuffer 50,menubuffer 100
180 :
190 PROCcolours
200 PROCsetup_sound
210 PROCinitwimp
220 PROCsetup_windows
230 :
240 ON ERROR PROCerrorbox
250 errorflag=FALSE
260 TIME=0:running=FALSE
270 REPEAT
280 WHILE running
290 beat=0:TIME=0
300 WHILE beat<beattotal
310 PROCplay:PROCgetnext
320 REPEAT:PROCpoll:UNTIL TIME>=speed
330 ENDWHILE
340 ENDWHILE
350 PROCpoll
360 UNTIL FALSE
370 :
380 DEF PROCpoll
390 SYS "Wimp_Poll",&30,block TO evnt%
400 CASE evnt% OF
410 WHEN 1:PROCredrawwindow(block!0)
420 WHEN 6:PROCselect(block!12,block!1
6)
430 WHEN 8:PROCkey
440 ENDCASE
450 ENDPROC
460 :
470 DEF PROCinitwimp
480 SYS "Wimp_Initialise"
490 SYS "Wimp_ForceRedraw",-1,0,0,1279
,1023
500 MOUSE TO 640,512:*POINTER
510 ENDPROC
520 :
530 DEF PROCsetup_sound
540 REM Omit next three lines for defa
ult voices
550 FOR voice=1 TO 32
560 SYS "Sound_RemoveVoice",0,voice
570 NEXT
580 RESTORE 100
590 FOR voice=1 TO 8
600 READ channel(voice)
610 NEXT voice
620 REM Omit the next line
630 REM for default voices
640 *RMLOAD PercusMod
650 PROCsetup_voices
660 ENDPROC
670 :
680 DEF PROCsetup_voices
690 FOR voice=1 TO 8
700 SYS "Sound_AttachVoice",voice,chan
nel(voice)
710 SYS "Sound_InstallVoice",0,channel
(voice) TO voice$(voice)
720 NEXT voice
730 VOICES 8
740 ENDPROC
750 :
760 DEF PROCsetup_windows
```

```
  770 iconstep=1280/beattotal
  780 iconwidth=iconstep-4
  790 FOR i%=1 TO 8
  800 top=1023-(i%*104)
  810 bottom=top-48
  820 window%(i%)=FNcreate_window(voice$
(i%),&81,black,white,1279,top,0,bottom,1
279,top)
  830 PROCcreate_icons(window%(i%),beatt
otal,0,0,bottom+4,0,iconwidth,40,iconste
p,&503D,white,black)
  840 PROCopen_window(window%(i%))
  850 NEXT i%
  860 window%(0)=FNcreate_window("",&80,
black,white,1279,1023,0,1000,1279,1023)
  870 PROCcreate_icons(window%(0),beatto
tal,0,0,1004,0,iconwidth,24,iconstep,&39
,white,black)
  880 PROCopen_window(window%(0))
  890 RESTORE 930
  900 FOR icon=0 TO 6
  910 READ $(menubuffer+(icon*10))
  920 NEXT
  930 DATA START,STOP,RESET,FASTER,SLOWE
R,SAVE,LOAD
  940 menubar=FNcreate_window("",&C0,blu
e,white,1199,100,80,0,1199,100)
  950 PROCcreate_icons(menubar,7,menubuf
fer,10,12,8,(1200/7)-40,80,1132/7,&213D,
cyan,blue)
  960 PROCopen_window(menubar)
  970 fname=FNcreate_window("Filename:",
&C1,black,white,1000,100,800,50,1000,100
)
  980 PROCcreate_icons(fname,1,fnbuffer,
11,54,8,300,40,0,&131,white,black)
  990 errorbox=FNcreate_window("ERROR",&
C1,white,red,700,500,300,300,700,500)
 1000 PROCcreate_icons(errorbox,1,errbuf
fer,30,400,50,300,48,0,&13D,white,blue)
 1010 ENDPROC
 1020 :
 1030 DEF PROCgetnext
 1040 FOR channel=1 TO 8
 1050 block!0=window%(channel)
 1060 block!4=beat
 1070 SYS "Wimp_GetIconState",,block
 1080 soundflag(channel)=(block!24) AND
(1<<21)
 1090 NEXT channel
 1100 block!0=window%(0)
 1110 IF beat<>0 THEN block!4-=1 ELSE bl
ock!4=beattotal-1
 1120 block!8=0:block!12=1<<21
```

```
 1130 SYS "Wimp_SetIconState",,block
 1140 block!4=beat:block!8=1<<21
 1150 SYS "Wimp_SetIconState",,block
 1160 beat+=1
 1170 ENDPROC
 1180 :
 1190 DEF PROCkey
 1200 IF block!0=fname AND block!24=&0D
fn=TRUE
 1210 ENDPROC
 1220 :
 1230 DEF PROCplay
 1240 FOR channel=1 TO 8
 1250 IFsoundflag(channel)<>0 THEN SOUND
 channel,-15,1,1
 1260 NEXT channel
 1270 TIME=0
 1280 ENDPROC
 1290 :
 1300 DEF PROCselect(window,icon)
 1310 CASE window OF
 1320 WHEN menubar AND NOT errorflag
 1330 CASE icon OF
 1340 WHEN 0:PROCstartplay
 1350 WHEN 1:PROCstop
 1360 WHEN 2:PROCreset
 1370 WHEN 3:speed+=speed<>0
 1380 WHEN 4:speed-=speed<>50
 1390 WHEN 5:PROCsave
 1400 WHEN 6:PROCload
 1410 ENDCASE
 1420 WHEN errorbox
 1430 errorflag=FALSE
 1440 ENDCASE
 1450 ENDPROC
 1460 :
 1470 DEF PROCstartplay
 1480 running=TRUE
 1490 PROCclearbeat(window%(0),beat)
 1500 beat=0
 1510 ENDPROC
 1520 :
 1530 DEF PROCstop
 1540 running=FALSE
 1550 PROCclearbeat(window%(0),beat)
 1560 beat=beattotal
 1570 ENDPROC
 1580 :
 1590 DEF PROCclearbeat(window,beat)
 1600 block!0=window
 1610 IF beat<>0 THEN block!4=beat-1 ELS
E block!4=beattotal-1
 1620 block!8=0:block!12=1<<21
 1630 SYS "Wimp_SetIconState",,block
 1640 ENDPROC
 1650 :
 1660 DEF PROCreset
 1670 PROCstop
 1680 FOR icon=1 TO beattotal
 1690 FOR window=1 TO 8
 1700 PROCclearbeat(window%(window),icon
)
 1710 NEXT window
 1720 NEXT icon
 1730 ENDPROC
 1740 :
 1750 DEF PROCsave
 1760 F%=OPENOUT(FNfilename)
 1770 PRINT#F%,speed,beattotal
 1780 FOR voice=1 TO 8
 1790 PRINT#F%,channel(voice)
 1800 NEXT
 1810 FOR channel=1 TO 8
 1820 FOR beat=0 TO beattotal-1
 1830 block!0=window%(channel)
 1840 block!4=beat
 1850 SYS "Wimp_GetIconState",,block
 1860 PRINT#F%,(block!24)
 1870 NEXT:NEXT
 1880 CLOSE#F%
 1890 ENDPROC
 1900 :
 1910 DEF PROCload
 1920 F%=OPENIN(FNfilename)
 1930 INPUT#F%,speed,beattotal
 1940 FOR voice=1 TO 8
 1950 INPUT#F%,channel(voice)
 1960 NEXT
 1970 PROCsetup_voices
 1980 PROCinitwimp
 1990 PROCsetup_windows
 2000 FOR channel=1 TO 8
 2010 FOR beat=0 TO beattotal-1
 2020 INPUT#F%,flags
 2030 block!0=window%(channel)
 2040 block!4=beat
 2050 block!8=flags AND 1<<21
 2060 block!12=0
 2070 SYS "Wimp_SetIconState",,block
 2080 NEXT:NEXT
 2090 CLOSE#F%
 2100 ENDPROC
 2110 :
 2120 DEF FNfilename
 2130 PROCstop:fn=FALSE
 2140 $fnbuffer=""
 2150 PROCopen_window(fname)
 2160 SYS "Wimp_SetCaretPosition",fname,
0,,,-1,-1
 2170 REPEAT:PROCpoll:UNTIL fn
 2180 SYS "Wimp_CloseWindow",,block
 2190 =$fnbuffer
 2200 :
 2210 DEF PROCopen_window(handle%)
 2220 block!0=handle%
 2230 SYS "Wimp_GetWindowState",0,block
 2240 SYS "Wimp_OpenWindow",0,block
 2250 ENDPROC
 2260 :
 2270 DEF PROCredrawwindow(handle%)
 2280 block!0=handle%
 2290 SYS "Wimp_RedrawWindow",0,block TO
 more%
 2300 WHILE more%
 2310 SYS "Wimp_GetRectangle",0,block TO
 more%
 2320 ENDWHILE
 2330 ENDPROC
 2340 :
 2350 DEF PROCcolours
 2360 black=0:red=1:green=2
 2370 yellow=3:blue=4:magenta=5
 2380 cyan=6:white=7:midgrey=15
 2390 scrollbarf=14
 2400 scrollbarb=14
 2410 highlightb=red
 2420 titlef=12
 2430 titleb=scrollbarf
 2440 VDU 19,0,24,128,128,128
 2450 VDU 19,15,16,128,128,128
 2460 VDU 19,14,16,15*16,11*16,6*16
 2470 VDU 19,13,16,0*16,12*16,15*16
 2480 VDU 19,12,16,0*16,0*16,8*16
 2490 VDU 19,11,16|
 2500 VDU 19,10,16|
 2510 VDU 19,9,16|
 2520 VDU 19,8,16|
 2530 ENDPROC
 2540 :
 2550 DEF FNcreate_window(title$,flags%,
fgcol%,bgcol%,maxx%,maxy%,wal%,wab%,war%
,wat%)
 2560 LOCAL handle%
 2570 block!0=wal%:block!4=wab%
 2580 block!8=war%:block!12=wat%
 2590 block!16=0:block!20=maxy%
 2600 block!24=-1:block!28=flags%
 2610 block?32=titlef:block?33=titleb
 2620 block?34=fgcol%:block?35=bgcol%
 2630 block?36=scrollbarb
 2640 block?37=scrollbarf
 2650 block?38=highlightb
 2660 block?39=0:block!40=0:block!44=0
 2670 block!48=maxx%:block!52=maxy%
 2680 block!56=&2D:block!60=&3000
 2690 $(block+72)=LEFT$(title$,11)
 2700 block!84=0
 2710 SYS "Wimp_CreateWindow",0,block TO
 handle%
 2720 =handle%
 2730 :
 2740 DEFPROCcreate_icons(window,number,
indirectbuffer,indtextlength,vpos,inset,
width height,separation,iconflags,bg,fg)
 2750 FORicon%=0 TO number-1
 2760 minx=icon%*separation+inset
 2770 maxx=minx+width
 2780 block!0=window:block!4=minx
 2790 block!8=vpos:block!12=maxx
 2800 block!16=vpos+height
 2810 block!20=iconflags
 2820 block?23=(bg<<4)+fg
 2830 IF (iconflags AND (1<<8))<>0 THEN
 2840 ictext=indirectbuffer+(icon%*indte
xtlength)
 2850 block!24=ictext
 2860 block!28=-1:block!32=11
 2870 ELSE
 2880 $(block+24)=""
 2890 ENDIF
 2900 SYS "Wimp_CreateIcon",,block
 2910 NEXT icon%
 2920 ENDPROC
 2930 :
 2940 DEF PROCerrorbox
 2950 LOCAL ERROR
 2960 ON ERROR PROCerror
 2970 PROCstop
 2980 errorflag=TRUE
 2990 $errbuffer=REPORT$
 3000 PROCopen_window(errorbox)
 3010 REPEAT PROCpoll
 3020 UNTIL errorflag=FALSE
 3030 block!0=errorbox
 3040 SYS "Wimp_CloseWindow",,block
 3050 block!0=fname
 3060 SYS "Wimp_CloseWindow",,block
 3070 ENDPROC
 3080 :
 3090 DEF PROCerror
 3100 MODE 12:*FX 4
 3110 *FX 221 1
 3120 *FX 225 1
 3130 *FX 200 0
 3140 REPORT:PRINT " at line ";ERL
 3150 END
```

MATRIX-3: A THREE-DIMENSIONAL SPREADSHEET

Matrix-3 is the third major new spreadsheet to be released for the Archimedes, and it exhibits some novel features. Mike Williams has been trying it out.

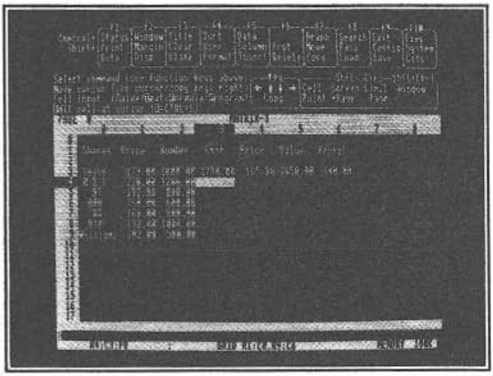
The Archimedes already benefits from the availability of a number of spreadsheet packages, of which the best known are probably Logistix (reviewed in RISC User Volume Issue 3) and SigmaSheet (reviewed in Volume Issue 7). Pipedream, though not specifically a dedicated spreadsheet program, also offers comparable facilities in conjunction with its word processing and database capabilities (reviewed in Volume Issue 8). Now Archimedes users have a further choice in the form of Matrix-3 from Cambridge Microsystems.

The packaging is indeed very smart. The 190 page manual is supplied in a ring binder with hard covers which slips into a matching case (more reminiscent of software on the PC or Apple Macintosh). A pocket at the rear of the manual holds the single disc and a keystrip.

Booting the disc quickly brings up a typical spreadsheet display consisting of rows and columns. In the case of Matrix-3 this is very logically designed for displaying and inputting information. There are also three levels of help, the default being the so-called 'novice' level. This provides a complete reminder of all the function and cursor key operations by which the software is controlled. Selecting any function within Matrix-3 causes a more detailed 'help' description to appear.

One of the more obvious and fundamental features of Matrix-3 is that it is 3-dimensional in form. A 'sheet' consists of rows and columns as usual (up to 10,000 in each case, but the real limitation is RAM). In addition, further layers or pages may be used (up to a maximum of 100). If you want to use Matrix-3 as a standard 'flat' spreadsheet then no further consideration of pages is required; the software just assumes all references are to the current page. But if your application needs extra pages, they are there just for the asking.

Cells may contain any one of four entities entered via the keyboard. Matrix-3 will automatically detect text or numeric input, while Ctrl-F and Ctrl-P are used to specify input of formulae or programs. In cases of ambiguity, text and numeric input can also be explicitly signalled. A formula, as you would expect, is a single statement, whereas a program can consist of many formulae and other statements.



One most useful feature is the automatic use of *cursor pointing*. Where a cell reference is required, moving the cursor to that cell causes the corresponding cell reference to be used when building up a formula or program. In addition, cells may be referred to by means of row, column and (if necessary) page numbers, or by user assigned row, column (and page) titles.

The heart of any spreadsheet lies in the facilities for creating, replicating and editing formulae, and (in Matrix-3) programs. All the usual arithmetic and logical operators are provided, and an extensive range of functions covering mathematical, statistical, general, matrix, and programming needs.

Cell contents can be replicated as required, either keeping cell references fixed or changing them relative to a new start position. However, I have yet to find a way of copying the result of a formula into another (or the same) cell, a facility which I find useful with other spreadsheets which I use. Any cell can be programmed to display the result of a formula, but that is not quite the same.

One of the most innovative features of Matrix-3 is the facility to create programs, and these really are like programs in Basic. A complete program, limited in length only by the availability of memory, can be created and stored in any cell. Matrix-3 programs can use five different data types:

string and numeric constants
cell references and pointers
variables

A cell reference is a standard reference to a cell location in terms of its row, column and page position.

Cell pointers, on the other hand, are variables used like indirection operators in Basic. If a cell pointer is initialised to a particular cell reference, then incrementing the cell pointer allows other cells to be accessed in turn. Incrementing may also use the letters R, C and P, followed by a number, to increment the cell pointer by the specified number of rows, columns or pages respectively. Variables are any user-defined identifier (maximum 8 characters) preceded by a '%' character.

Nine different types of statement are possible in Matrix-3. These are:

assignment
goto
call
while-do
comment
pointer
conditional
return
cell assignment

Most of these have their Basic equivalents. The pointer statement simply assigns to a cell pointer the cell reference. The conditional statement is that old friend, the IF-THEN-ELSE construction. Cell assignment allows the result of an expression to be assigned to a cell.

Call and return provide a simple subroutine facility. The call statement specifies a label marking the start of the subroutine. A much more useful facility is the availability of an EVAL function. This takes as its argument a cell reference, and when called will execute any formula or program stored in that cell. This provides the equivalent of Basic's procedure handling, though no direct parameter passing is possible. Values can be passed using other cells.

I have dwelt at some length within this short review on the programming capability of Matrix-3, but it does strike me as a particularly powerful and flexible addition to what most spreadsheets have to offer.

Regrettably there is no space to give more than the briefest mention to several other features of Matrix-3. Display formats can also be defined by the user. The screen can be split vertically or horizontally into two separate windows for viewing different parts of a sheet. Rows or columns can be sorted into ascending or descending order. There are also some relatively elementary graph-drawing facilities covering bar, line and point charts, and future versions of Matrix-3 will allow data to be exported to *Presenter* (also reviewed in this issue) for better presentation.

The documentation is well produced, but I felt that it was somewhat dry and academic in style. Some example files are included on the disc, but there is very little description of these in the manual, nor any other worked examples of spreadsheets.

There are many good features to Matrix-3, not least the help screens. However, I have to confess that I found neither the software nor the manual quite as helpful as I would have liked when setting myself some realistic tasks. The programming facility is very attractive but I found some initial problems in using this feature correctly and there was no explanation in the manual of the error message I encountered.

Matrix-3 is a good product, well worth the attention of anyone seeking a spreadsheet on the Archimedes. Personally, I still prefer Acorn's Logistix for ease of use plus a wide range of features, but Matrix-3 is a strong challenger in this field.

Product	**Matrix-3**
Supplier	**Cambridge Microsystems** **19 Panton Street,** **Cambridge CB2 1HL.** **Tel. (0223) 66553**
Price	**£109.25 inc. VAT.**

A REAL-TIME IMAGE SPINNER (2)

Stephen Streater adds some new features to last month's real-time image spinner.

The program needs 160K of screen RAM and 270K of User RAM.

The Image Spinner program published last month allowed you to create a variety of visual effects in real time. This month's listing provides a number of useful additions. With the new version you can operate on any rectangular part of an image (last month's would only spin a whole screen). The new version will also allow any scaling factor to be applied to the image, so that you can magnify the original as well as reduce it. The new routine also allows you to hold a number of images in memory at the same time, so that you can simultaneously spin more than one image. Code has also been incorporated to automatically clip images extending beyond the edge of the screen, and to cope with rotation angles outside of the range 0 to 2 pi.

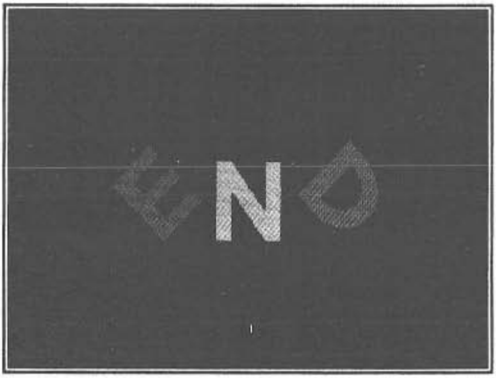
WHAT IT CAN DO

The extent of the enhancements explains, I hope, why the new bits of code are lengthy. But with these additions, the routines become much more powerful, and can be used for many purposes other than just spinning images. Essentially, the program allows you to "grab" any part of a screen image, and place it back on the screen at any magnification or reduction, at any position, and at any angle. By repeating the procedure within a loop, you can create a wide variety of effects.

GETTING IT GOING

To make the program operational, you need to incorporate this month's code into last month's program. It is vitally important to keep to the line numbers as published (both magazine and disc versions of the original program used the same line numbering). The listing published here will work as an EXEC file, so if you have a text editor, you may care to type it in, save away the text file, and then EXEC this into last month's program. If not, you should load in last month's program, and type in the amendments very carefully. Note the two deletion sequences. Once you have done this, save the new program before running it.

When you run the new program, you will see the word "END" appear in bold coloured lettering during set up. Then the screen will clear, and the animation will begin. The "E" spirals in from the upper left part of the screen, the "D" from the upper right, and the "N" is elevated from the middle. All this occurs simultaneously, and needless to say the three letters finish up in the correct position before the sequence repeats.



EXPERIMENTING WITH THE PROGRAM

The program is easy to alter in order to create other effects. To try it out, press Escape, and enter:

```
PROCrot(640,512,200,300,0,1)
```

This will place a large "N" on the screen. The position of its centre is determined by the first two parameters, and its size by the next pair. Its angle of rotation is given by the fifth parameter, and although we have used 0, you could use any value, though remember that it is in radians, not degrees. The final parameter is a new one. It specifies which bank the image is to be taken from. Bank 0 holds the "E", bank 1 the "N", and bank 2 the "D".

If you are going to use the program with different images, there are two further procedures which you will need to call. PROCtidy should be called once at the start. It has a single parameter, the number of image banks which will be used to store images (3 in this case). PROCtidy also assembles the whole machine code program. Secondly you will need PROCscreen. This copies any part of the main screen into an image bank, magnifying it to fill the whole bank. It takes five parameters: the x and y co-ordinates of the centre of the rectangle to be grabbed, the width and height of the rectangle to be grabbed, and the bank number into which the image is to be placed. In the "END" example, we have called this procedure three times to store each of the three letters in its own bank. The three letters were drawn on the screen using ordinary graphics commands, but you could equally well load in one or more screens or sprites instead.

Just one more point: the example uses screen flipping to keep the animation as smooth as possible. In other words, each new screen is created when the user is viewing the previous one. This is achieved using OS Byte &70 and &71 to switch banks (screen banks, not image banks) before each new screen is created. Shadow bank switching is particularly useful when you need to erase the image on the previous frame before creating the next, because of the extra time which this takes, and the flicker caused by the clearing operation. In our example, when the "E" and the "D" spiral in they leave no trail because previous frames are cleared. In last month's example, we did not resort to bank switching because we could leave all past images on the screen, and simply overlay the new ones.

With a bit of experimenting, you should be able to create some interesting effects. If you do, we would like to hear from you.

```
   10 REM          >Spinner
   30 REM Version   A 1.2
   75 ON ERROR MODE 13:REPORT:PRINT" at
line ";ERL:END
   80 PROCtidy(3)
   90 REM Example starts here
  100 MODE 15:MODE 13:OFF:bcg=18:bcgt=0
  104 COLOUR bcg+128:CLS
  110 GCOL41:RECTANGLE FILL 64,64,192,32
0
  114 GCOL15:RECTANGLE FILL 320,64,256,3
20
  116 GCOL28:RECTANGLE FILL 640,64,64,32
0
  118 MOVE 704,224:MOVE 704,64
  120 PLOT &B5,704,384:MOVE 384,64
  122 GCOL bcg TINT bcgt:MOVE 384,256
  124 PLOT &55,512,64:MOVE 384,384
  126 MOVE 512,384:PLOT &55,512,192
  128 RECTANGLE FILL 128,128,128,64
  130 RECTANGLE FILL 128,256,128,64
  132 MOVE 704,224:MOVE 704,128
  134 PLOT &B5,704,320
  136 PROCscreen(160,224,192,320,0)
  138 PROCscreen(448,224,256,320,1)
  140 PROCscreen(768,224,256,320,2)
  141 REPEAT
  142 video%=1
  144 FOR s=0 TO 250 STEP 5
  146 SYS "OS_Byte",&70,video%
  148 SYS "OS_Byte",&71,3-video%
  150 video%=3-video%:WAIT:CLS
  152 screen_adr = FNfind_screen
  154 PROCrot(90+s,512,s*3/4,s,s/25-10,0
)
  156 PROCrot(600,262+s,200,s,0,1)
  158 PROCrot(1150-s,512,s,s,10-s/25,2)
  160 NEXT
  162 SYS "OS_Byte",&71,1
  164 zz=INKEY(500)
  166 UNTIL FALSE
  190 b = 0:[OPT Z
DELETE 200,360
  400     CMP   R0, #320<<16:BLT e_1
 1251 .input   EQUD 148:EQUD TRUE
 1252 .output  EQUD 0
 1253 .screen ADR   R0, input
 1254     ADR   R1, output
 1255    SWI    "OS_ReadVduVariables"
 1256    MOV    PC, R14
 1257 .x_begin EQUD 0:.y_begin EQUD 0
 1258 .x1 EQUD 0:.y1 EQUD 0
 1259 .x2 EQUD 0:.y2 EQUD 0
 1260 .x3 EQUD 0:.x4 EQUD 0
 1261 .temp  EQUD 0:EQUD 0:EQUD 0:EQUD 0
 1262 .t_    EQUD temp
 1263 .link  EQUD 0:.stack EQUD 0
 1264 .workspace1 EQUD workspace
 1265 .resize_3:FN_init_resize
DELETE 2170,2640
 2720 DEF PROCrot(A%,B%,C%,D%,t,bank)
 2721 IF bank>=0 THEN
 2722 !workspace1=workspace+80*1024*bank
 2723 !output=screen_adr
 2724 ENDIF
 2725 WHILE t<0:t+=2*PI:ENDWHILE
 2726 WHILE t>2*PI:t-=2*PI:ENDWHILE
 2727 IF C%<8 THEN C%=8
 2728 IF D%<8 THEN D%=8
 2740 r=SQR(C%*C%+D%*D%)/8:B%=1023-B%
 2830 IF SIN(t)*xs<1 THEN
 2840    !x_begin=z*(A%/4-r*COS(a))
 2850    !y_begin=z*(B%/4+r*SIN(a))
 2860    !x1=z/xs:!y1=0:!y2=z/ys
 2870    !x3=0:!x4=z*64*256:!x2=z*64*256
 2880 ELSE
 2890    IF COS(t)*ys<1 THEN
 2900    !x_begin=z*(A%/4-r*COS(a-PI/2))
 2910    !y_begin=z*(B%/4+r*SIN(a-PI/2))
 2920    !x1=0:!y1=z/ys:!y2=0
 2930    !x3=z*64*256:!x4=0:!x2=z/xs
 2940    ELSE
 2950    !x_begin=z*(A%/4-r*COS(a-t))
 2960    !y_begin=z*(B%/4+r*SIN(a-t))
 2970    !x1=z*COS(t)/xs:!y1=z*SIN(t)/ys
 2980    !y2=z/COS(t)/ys:!x3=z*TAN(t)
 2990    !x4=z/TAN(t):!x2=z/SIN(t)/xs
 3000    ENDIF
 3010 ENDIF
 3020 ENDPROC
12980 :
12990 DEF PROCset_up2(t)
13000 IF SIN(t)*xs<1 THEN
13010    !x_begin=z*(A%/4+r*SIN(b-PI))
13020    !y_begin=z*(B%/4+r*COS(b-PI))
13030    !x1=z/xs:!y1=0:!x2=z*64*256
13040    !x3=-z*64*256:!y2=z/ys:!x4=0
13050 ELSE
13060    IF COS(t)*ys>-1 THEN
13070    !x_begin=z*(A%/4+r*SIN(b-PI/2))
13080    !y_begin=z*(B%/4+r*COS(b-PI/2))
13090    !x1=0:!y1=z/ys:!x2=z/xs
13100    !x3=0:!y2=-z<<14:!x4=-z<<14
13110    ELSE
13120    !x_begin=z*(A%/4+r*SIN(b-t))
13130    !y_begin=z*(B%/4+r*COS(b-t))
13140    !x1=-z*COS(t)/xs:!y1=z*SIN(t)/y
s
13150    !x2=z/SIN(t)/xs:!x3=-z/TAN(t)
13160    !y2=-z/COS(t)/ys:!x4=-z*TAN(t)
13170    ENDIF
13180 ENDIF
13190 ENDPROC
13200 :
13210 DEF FN_plot(b)
13220 [OPT Z:MOV   R4, R1, LSR #16
13230    ADD   R4, R4, R4, LSL #2
13240    ADD   R4, R9, R4, LSL #6
13250    LDRB  R4, [R4, R0, LSR #16]
13260    STRB  R4, [R12, R2, LSR #16]
13270    ADDS  R2, R2, R10:]
13280 IF a THEN
13290    [OPT Z:cmp   r2, #320<<16
13300    bpl   fos(b):]
13310 ELSE
13320 [OPT Z:bmi fos(b):]
13330 ENDIF:=0
13340 :
13350 DEF FN_init_loop(x,y)
13360 b += 1: IF b=10 THEN
13380    [OPT Z:B      memory_2
13400 .next_0 ADDS  R0, R0, R6, LSL #8
13420    ADD   R1, R1, R7, LSL #8
13430    ADD   R2, R2, R10, LSL #8
13440    BLT   next_1
13450    CMP   R0, #320<<16:BGE next_1
13460    CMP   R1, #0:BLT next_1
13470    CMP   R1, #256<<16:BLT next_0
13480 .next_1 SUB   R0, R0, R6, LSL #8
13490    SUB   R1, R1, R7, LSL #8
13500    SUB   R2, R2, R10, LSL #8
13510    MOV   R11, #7
13520 .next_ ADDS  R0, R0, R6, LSL R11
13540    ADD   R1, R1, R7, LSL R11
13550    ADD   R2, R2, R10, LSL R11
13560    BLT   next_2
13570    CMP   R0, #320<<16:BGE next_2
13580    CMP   R1, #0:BLT next_2
13590    CMP   R1, #256<<16:BLT next_
13600 .next_2 SUB   R0, R0, R6, LSL R11
13610    SUB   R1, R1, R7, LSL R11
13620    SUB   R2, R2, R10, LSL R11
13630 .next_2a SUBS  R11, R11, #1
13640    BGT   next_
13650 .next_8a ADDS  R0, R0, R6
13660    ADD   R1, R1, R7
13670    ADD   R2, R2, R10
13680    BLT   next_9
13690    CMP   R0, #320<<16:BGE next_9
13700    CMP   R1, #0<<16:BLT next_9
13710    CMP   R1, #256<<16:BLT next_8a
13720 .next_9 LDR   R9, memory
13730    LDMIA R9, {R9-R12}:MOV R15, R14
13740 .memory EQUD  memory_0
13750 .memory_0 EQUD0:EQUD0:EQUD0:EQUD0
13760 .memory_1 EQUD  0:.memory_2:]
13770 ENDIF:[OPT Z
13780    LDR R0,memory:STMIA R0, {R9-R12
}
13790    MOVS  R0, R9:MOVMI R0, #0
13800    MOVS  R1, R10:MOVMI R1, #0
13810    MOV   R2, R11:MOV R3, R12:]
13820 IF a THEN
13830    [OPT Z:CMP R2, #320<<16
13840    MOVPL R10, #1<<16:BPL fos(b)
13860    CMP   R2, #0:RSBMI R2, R2, #0
13870    MOVMI R2, R2, LSR #16
13880  MULMI R9, R6, R2:MULMI R10, R7, R
2
13890    MOVMI R2, #0:]
13900    IF x THEN
13910    [OPT Z:ADDMI R0, R0, R9:]
13920    ELSE
13930    [OPT Z:SUBMI R0, R0, R9:]
13940    ENDIF
13950    IF y THEN
13960    [OPT Z:ADDMI R1, R1, R10
13970    MOV   R10, #1<<16:]
13980   ELSE
13990   [OPT Z:SUBMI R1, R1, R10
14000    MOV   R10, #1<<16:]
14010    ENDIF
14020 ELSE
14030    [OPT Z:CMP R2, #0
14040    MOVMI R10, #&FF000000
14050    ADDMI R10, R10, R10, LSR #8
14060    BMI   fos(b)
14070    CMP   R2, #320<<16
14080    SUBPL R2, R2, #320<<16
14090    ADDPL R2, R2, #1<<16
14100    MOVPL R2, R2, LSR #16
14110 MULPL R9, R6, R2:MULPL R10, R7, R2
14120    MOVPL R2, #320<<16
14130    SUBPL R2, R2, #1<<16:]
14140    IF x THEN
14150    [OPT Z:ADDPL R0, R0, R9:]
14160    ELSE
14170    [OPT Z:SUBPL R0, R0, R9:]
14180    ENDJF
14190    IF y THEN
14200    [OPT Z:ADDPL R1, R1, R10
14210    MOV   R10, #&FF000000
14220    ADD   R10, R10, R10, LSR #8:]
14230    ELSE
14240    [OPT Z.SUBPL R1, R1, R10
14250    MOV   R10, #&FF000000
14260    ADD   R10, R10, R10, LSR #8:]
14270    ENDIF
14280 ENDIF
14290 [OPT Z
14300    RSBS    R9, R0, #320<<16
14310    RSBPLS R9, R1, #256<<16
14320    CMPPL  R0, #0:CMPPL  R1, #0
14330    BMI    fos(b)
14340    CMP    R3, #0<<16:BMI fos(b)
14350    CMP    R3, #256<<16:BPL fos(b)
14360    LDR    R9, workspace1
14370    MOV    R12, R3, LSR #16
14380    ADD    R12, R12, R12, LSL #2
14390    ADD    R12, R13, R12, LSL #6
14400 ]:=0
14410 :
14420 DEF FN_r_A(a)
14430 [OPTZ:FN_init_loop(0,1)
14440 .repeat FN_plot(b)
14450    SUBS  R0, R0, R6
14460    ADD   R1, R1, R7
14470    ADDLT R15, R15, #4
14480    CMP   R1, #256<<16:BLT repeat
14490    LDR   R9, memory
14500    LDMIA R9, {R9-R12}
14510    B     off_screen(b)
14520 .fos(b) RSB R6, R6, #0
14530    STR   R14, memory_1
14540    BL    next_0
14550    LDR   R14, memory_1
14560    RSB   R6, R6, #0
14570 .off_screen(b)
14580 ]:=0
14590 :
14600 DEF FN_r_B(a)
14610 [OPTZ:FN_init_loop(1,0)
14620 .repeat FN_plot(b)
14630    ADD   R0, R0, R6
14640    SUBS  R1, R1, R7
14650    ADDLT R15, R15, #4
14660    CMP   R0, #320<<16:BLT repeat
14670    LDR   R9, memory
14680    LDMIA R9, {R9-R12}
14690    B     off_screen(b)
14700 .fos(b) RSB   R7, R7, #0
14710 STR  R14, memory_1:BL next_0
14720 LDR  R14, memory_1:RSB R7, R7, #0
14730 .off_screen(b)
14740 ]:=0
14750 :
14760 DEF FN_r_C(a):[OPTZ
14770    FN_init_loop(1,1)
14780 .repeat FN_plot(b)
14790    ADD   R0, R0, R6
14800    ADD   R1, R1, R7
14810 CMP R0,#320<<16:ADDGE R15,R15,#4
```

```
14820    CMP   R1, #256<<16:BLT repeat
14830    LDR   R9, memory
14840    LDMIA R9, {R9-R12}
14850    B     off_screen(b)
14860 .fos(b) STR R14, memory_1
14870    BL    next_0:LDR R14, memory_1
14880 .off_screen(b)
14890 ]:=0
14900 :
14910 DEF FN_r_D(a):[OPTZ
14920    FN_init_loop(0,0)
14930 .repeat FN_plot(b)
14940    SUB   R0, R0, R6
14950    SUBS  R1, R1, R7
14960    ADDLT R15, R15, #4
14970    CMP   R0, #0:BGE repeat
14980    LDR   R9, memory
14990    LDMIA R9, {R9-R12}
15000    B     off_screen(b)
15010 .fos(b) RSB R6, R6, #0
15020 RSB  R7, R7, #0:STR R14, memory_1
15030    BL    next_0:LDR R14, memory_1
15040    RSB  R6, R6, #0:RSB R7, R7, #0
15050 .off_screen(b)
15060 ]:=0
15070 :
15080 DEF PROCscreen(x_centre,y_centre,x
_size,y_size,bank_no)
15090 !output = workspace+80*1024*bank_n
o
15100 !workspace1 = screen_adr
15110 PROCrot(640-(x_centre-640)*1280/x_
size,512-(y_centre-512)*1024/y_size,1280
*1280/x_size,1024*1024/y_size,0,-1)
15120 ENDPROC
15130 :
15140 DEF PROCtidy(no_of_banks)
15150 DIM off_screen(20), fos(20)
15160 DIM s% 8000+80*1024*no_of_banks:CL
S
15170 PROCasm(0,s%):PROCasm(2,s%)
15180 screen_adr=FNfind_screen
15190 ENDPROC
15200 :
15210 DEF FNfind_screen:CALL screen
15230 =!output
```

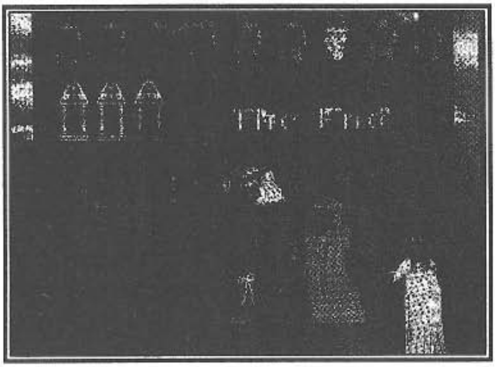
Archimedes Visuals

This month's Visuals are both from Julian Mudd.

the first program needs 160K of screen RAM

1 Sizzling Cross-Fade

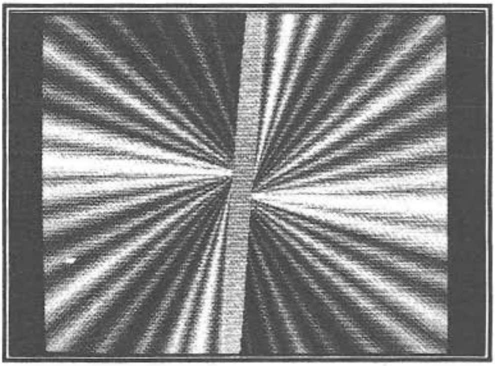
This program runs a carousel of screens, cross-fading between each. We have published cross-faders in earlier issues, but this one creates a quite different effect. Instead of individual sectors of the screen swapping to the new image one by one until the change-over is complete, here a pixellated rippling effect is created in which large areas of the screen seem to hover in a state which simultaneously reflects aspects of both new and old images. This rippling continues for some time with subtle palette changes until the new screen finally clarifies.



The effect is achieved by continually combining pixel information from the two screens and displaying the result. The routine works for any 80K screen, and obtains screen addresses legally, so that it will work equally well on 300 and 400 series machines. You will however need at least 160K of screen RAM, since shadow RAM is used. Finally a note about the screen images used by the program. They must be created using *SAVE as described in RISC User Volume 1 Issue 3 page 9, though no palette information is needed. The names of the image files appear in DATA statements in lines 200 and 210 of the program.

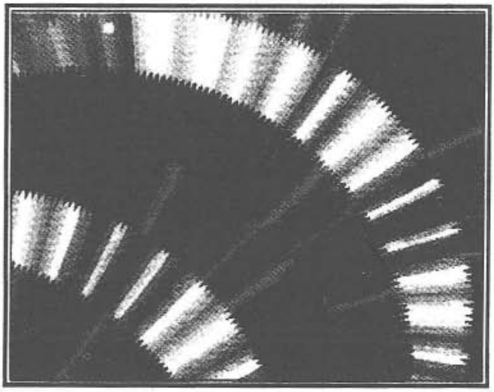
```
   10 REM          >Sizzle
   20 REM Program  Pixellated Fade
   30 REM Version  A 0.3
   40 REM Author   J.H.Mudd
   50 REM RISC User November 1988
   60 REM Program  Subject to Copyright
   70 :
   80 MODE 13:OFF
   90 PROCscrnaddr
  100 PROCassemble
  110 :
  120 REPEAT
  130 RESTORE
  140 REPEAT
  150 READ name$
  160 IF name$<>"End" OSCLI("LOAD "+name
$+" "+STR$~scrn2):CALL swap
  170 UNTIL name$="End"
  180 UNTIL FALSE
  190 :
  200 DATA Fractal1,Droom,Fractal2
  210 DATA Droomend,Fractal3,SprayPIC
  220 DATA End
  230 :
  240 DEFPROCscrnaddr
  250 DIM buff% &30
  260 !buff%=148
  270 buff%!4=7
  280 buff%!8=-1
  290 SYS "OS_ReadVduVariables",buff%,bu
ff%+&10
  300 scrn1=buff%!(&10)
  310 scrn2=scrn1+buff%!(&14)
  320 ENDPROC
  330 :
  340 DEF PROCassemble
  350 DIM code 1000
  360 pixelcount=0:byte1=1
  370 byte2=2:nibble1=3
  380 nibble2=4:destination=5
  390 source=6:maxaddress=7
  400 link=14
  410 FOR pass=0 TO 2 STEP 2
  420 P%=code
  430 [OPT pass
  440 .swap    LDR   destination,screen1
  450 LDR   source,screen2
  460 MOV    maxaddress,source
  470 MOV    pixelcount,#&140000
  480 .next     LDRB  byte1,[destination]
  490 MOV    nibble1,byte1,LSR #4
  500 AND    byte1,byte1,#&0F
  510 LDRB   byte2,[source]
  520 MOV    nibble2,byte2,LSR #4
  530 AND    byte2,byte2,#&0F
  540 CMP    byte1,byte2
  550 ADDLO byte1,byte1,#1
  560 SUBHI byte1,byte1,#1
  570 CMP    nibble1,nibble2
  580 ADDLO nibble1,nibble1,#1
  590 SUBHI nibble1,nibble1,#1
  600 ADD    byte1,byte1,nibble1,LSL #4
  610 STRB   byte1,[destination]
  620 ADD    destination,destination,#&E7
  630 ADD    source,source,#&E7
  640 CMP    destination,maxaddress
  650 SUBHS destination,destination,#&14
000
  660 SUBHS source,source,#&14000
  670 SUBS   pixelcount,pixelcount,#1
  680 BNE    next
  690 .exit     MOV    PC,link
  700 .screen1 EQUD  scrn1
  710 .screen2 EQUD  scrn2
  720 ]:NEXT
  730 ENDPROC
```

Colour Spinner



This very short program uses a narrow spinning ellipse to spread the 256 colour palette around the screen. For an interesting variant, try changing the last two lines to:

```
210 ELLIPSE FILL 640-N%,512-N%,32,
   840,PI*N%/160
220 UNTIL N%>1100
```



Both versions of the program are included on the magazine disc.

```
 10 REM              >Swirl
 20 REM Program      Colour Spinner
 30 REM Version      A 0.2
 40 REM Author       J.H.Mudd
 50 REM RISC User November 1988
 60 REM Program      Subject to Copyright
 70 :
 80 MODE 13:OFF
 90 DIM C%(7)
100 C%(0)=0:C%(1)=1:C%(2)=2:C%(3)=3
110 C%(4)=3:C%(5)=2:C%(6)=1:C%(7)=0
120 N%=0
130 :
140 REPEAT
150 N%+=1
160 A%=C%(N% MOD 8)
170 B%=C%((N% DIV 4)MOD 8)
180 C%=C%((N% DIV 16)MOD 8)
190 D%=C%((N% DIV 64)MOD 8)
200 GCOL C%+(D%<<2)+(B%<<4) TINT A%<<6
210 ELLIPSE FILL 640,512,32,840,PI*N%/
160
220 UNTIL FALSE
```

DABS
PRESS

Dabhand User News

Dabs Press are pleased to announce the launch of **ABC – the Archimedes Basic Compiler** the fast and powerful way to write instant machine code!

ABC takes programs written in BASIC V and transforms them into super-fast ARM machine code at a snap and ready for immediate action. No more are you limited by the time consuming interpretive process, or the need to be 'in' BASIC to run your programs. A simple * command is all that is required after passing through ABC.

But the power of ABC goes much further – by including compiler directives it is possible to control the manner in which ABC compiles programs. For example, by placing just *six* simple REM statements at the start of the BASIC program ABC will automatically compile any BASIC program into a fully relocatable module for loading into the RMA and which can be used from any enviroment.

These are just some of the features of ABC a program which can give you up to a 4000% speed increase.

ABC was written by Paul Fellows – ex-Acorn Computers and head of the team which wrote the Arthur Operating System and is excellent value at just £99.95 inclusive. Do we need to say any more?

Archimedes Basic Compiler

THE FAST AND POWERFUL WAY TO WRITE MACHINE CODE ACHIEVING SPEED INCREASES OF UP TO AND OVER 4000%!

The Archimedes Basic Compiler makes writing machine code programs and relocatable modules as easy as ABC! Programs can be developed and written using the full range of BASIC V's error messages and reports and once fully working, run through ABC which transforms them into machine code, ready for immediate action.

ABC is very easy to use and makes full use of Archimedes windows. Programs may be compiled from RAM or disc or using any combination of RAM and disc.

Just look at some of the advantages of using ABC:

- **Speed**: Compiled programs may run considerably faster than BASIC ones. As the benchmarks below show speed increases of up to 4000% are possible!
- **Syntax errors:** Once a program has been compiled you can be sure that it is free of syntax errors. With the interpreter there is no guarantee of this.
- **Stand alone code**: The object program which is produced by the compiler is entirely "stand-alone" and could for example be used as a utility or library package by another application program. You can re-run the program or module later *without* either ABC or ROM BASIC being present.
- **Language Specification:** The compiler accepts BBC BASIC programs in their standard tokenised form making it possible to compile many programs directly with little or no modification.
- **Data Types:** Variables and arrays with up to eight dimensions of any of the three basic data types, integer, floating point and strings are supported.
- **Assembler:** The compiler allows use of the in-line assembler within programs.
- **Compiler Directives:** A wide range of compiler directives are built into ABC including simple commands which allow modules to be compiled. These directives are built into REM statements so that the modules may be fully tested in BASIC first.

Sample Benchmarks

Benchmark	BASIC	ABC	Performance
GRAFSCRN	1.68	0.84	200%
INTMATH	0.19	0.02	950%
SIEVE – 1651 primes	5.16	0.58	890%
TAK(18,12,6)	27.53	0.78	3530%
FIBONACCI	49.43	1.40	3531%
ACKERMAN(3,4)	4.89	0.12	4075%
INT-ARRAY	1.84	0.34	541%
WHILE	13.11	0.40	3278%
REPEAT-UNTIL	12.75	0.37	3446%
FOR-NEXT	2.15	0.29	741%

ABC is supplied with two discs including a disc full of example programs and two manuals – a 150 page Reference Guide and User Guide. The price is just £99.95 inclusive of VAT and postage and packing. **See us at The Micro User Show!**

Archimedes Assembly Language: 368 pages devoted to programming the Archimedes in machine code. At £14.95 this is the *only* book which deals specifically with assembler on the Archimedes. Book and programs disc £21.95.

C: A Dabhand Guide. The definitive guide to C on your Archie. 512 pages just £14.95.

Free Catalogue and Ordering Details

Write to us, phone us, or send us a mailbox and we will send you, free of charge, our information packed catalogue giving full details of all our products.

Send cheques, POs, official orders to the address below, or quote your Access/Visa card number and expiry date. Credit card orders accepted by phone, letter or mailbox. P&P free in UK/BFPO. Elsewhere add £2.50 or £12 airmail.

Dabs Press (RUN), 5 Victoria Lane Whitefield, Manchester, M25 6AL.
Phone: 061-766-8423 Prestel: 942876210 BT Gold: 72:MAG11596

ANIMATING ARCHIE (Part 4)

This month Lee Calcraft looks at Using Delta Files.

The program needs 160K of screen RAM, and up to 250K of user RAM (see text).

DELTA FILES

The Delta File technique is ideally suited to saving and displaying animation screens. It works by taking advantage of the massive redundancy in most screens in any animated sequence. The principle involved is very simple. Instead of displaying a whole screen for each new frame, all you need to do is to display the differences between the new screen and the previous one - hence the name Delta. Thus a file containing the data for a whole frame of an 80 or 160K screen might only be a few bytes in length. The advantages of using such a system are three-fold. It uses less disc space to save each individual frame, less space in RAM, and screen writing can be performed at great speed, because only a small part of the screen will in most instances be involved.

This is exactly the technique used in Acorn's famous "Molecule" animation. Of the 50 or so disc files which are used in the animation, only one is a full 80K screen dump. The remainder are Delta Files. Each of these contains information on any changes between the frame to which it refers, and the previous one. If the frames had been treated in the normal way as straight screen files, the display would have needed 4M bytes of storage, both on disc, and in RAM. The degree of saving of course depends entirely on the subject matter, and in many cases it is possible to create sequences measured in hundreds rather than tens of frames on a 1M byte Arc.

The code for handling Delta Files is really very simple, although it must be written in ARM assembler in order to achieve the necessary speed. To create a set of Delta Files on an Arc we could use the following sequence of operations:

1. Create the first screen of the sequence.
2. Save the whole screen to disc.
3. Copy it to shadow RAM.
4. Draw the second frame on screen.
5. Create the first Delta File, based on the difference between the displayed screen and the shadow screen.
6. Copy the displayed screen to shadow RAM.
7. Draw the third frame on screen.

And so on.

To replay the sequence, all we need to do is to load in the first screen, then for each new frame, simply overlay the differences stored in the corresponding Delta File. To save the sequence, we can either save each file separately, as Acorn have done with the "Molecule", or save the whole data in a single disc file.

AN ARC IMPLEMENTATION

The accompanying program implements a Delta File system on the Arc. The example which it contains uses shadow screens in mode 12, and thus requires 160K of screen RAM (use *Configure ScreenSize 20 on a 300 series), and some 250K of user RAM. However, it needs no sprite space, so if you are using a 305, you could configure sprite space to zero. In any event, if you do not have enough user RAM, just reduce the size of bsize% in line 100, and the program will compensate by ending the sequence early. Moreover, the program is mode independent, and can easily be adapted to use a screen mode of more modest memory requirements.

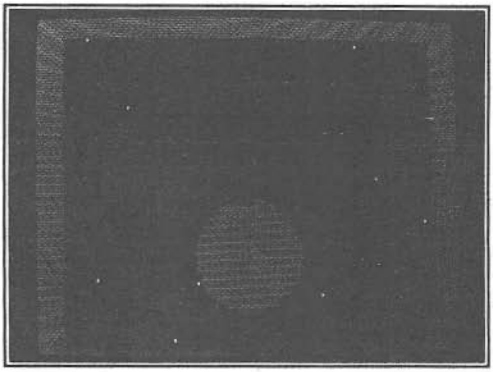
The heart of the program is a short piece of assembler which contains four separate routines: one to initialise, one to copy screens to the shadow screen, one to create Delta Files, and one to display them. The program also contains a demonstration to show how the system works. If you run the program, the code will be assembled, then you will see a sequence of 156 screens created in around one minute. As the display indicates, this uses about 185K of RAM. To store these screens conventionally would have taken over 12M bytes of RAM.

Once the sequence of files has been created in RAM, pressing the space bar will start the display, while pressing any other key will stop the program and prompt you to save the assembled machine code for use next month. If you opt for the space bar, you should see a green planet recede from you against a starry blue sky. The display is perfectly smooth, and you will notice that background stars are "uncovered" as the planet recedes. This would have been difficult to organise using the sprite techniques discussed in earlier issues. If you time it, you will see that the display lasts for around seven seconds. This is achieved by using a double WAIT statement in the display routine. This gives a display rate of 25 frames per second. If you remove one of the WAIT statements, you will double the display rate, and halve the display time.

LIMITATIONS

The receding planet example used in the program could have been made more complex without significantly increasing the amount of RAM used. We could have put some detail on the planet's surface, and could have arranged for some of the "stars" to recede at the same rate as the planet. And other objects could also have been introduced. There are in fact few limitations to the use of this method. You obviously cannot run the sequence backwards without creating a new set of Delta Files for the purpose, but this will rarely be a problem.

The only real restriction is on the degree of change from one screen to the next. There is in fact a break-even point at 50% of the screen size. Each difference recorded by the program takes two words of memory: one to give the screen location of the difference, and the other to give the screen word situated at that point. In other words the Delta File consists of a sequence of word pairs, the first giving a screen offset, the second the pixel data. In a normal screen save, the screen is stored as a sequence of pixel data held in 32 bit words. No screen reference points are required, since it is assumed that the first word refers to the start of the screen, and so on.



CREATING YOUR OWN SEQUENCE

It is an easy matter to create your own animation sequence, using the Delta File code supplied here. First you will need to replace the definition of PROCbackground with one of your own. The one in the example just clears the screen to blue, and displays a set of stars. Then you need to replace the three CIRCLE FILL statements with code to draw the objects to be animated. Note that two of these statements draw a circle of fixed size, while the third is used in a REPEAT loop to generate the full sequence.

Finally, here are some technical notes on the machine code. You should only need this if you are going to use the Delta File generator in a different framework from that supplied. The code has four entry points corresponding to four separate routines:

code	**Initialise screen parameters**
code+4	**Copy screen to shadow**
code+8	**Create next Delta File**
code+12	**Display next Delta File**

The last two of these must be entered with A% holding the start address of the next Delta File in RAM. On exit these two routines return a pointer to the next file, or zero if the buffer has been exceeded.

```
   10 REM            DeltaFile
   20 REM Program   Deltafile
   30 REM Version   A 0.9G
   40 REM Author    Lee Calcraft
   50 REM RISC User November 1988
   60 REM Program   Subject to Copyright
   70 :
   80 MODE12
   90 DIM buff &30,code &200
  100 bsize%=&30000:DIM screens bsize%
  110 PROCassemble
  120 :
  130 MODE12:OFF
  140 CALL code   :REM initialise
  150 REM-------------------------------
  160 REM           Create Files
  170 PROCbackground
  180 CIRCLE FILL 640,-200,640
  190 CALL code+4 :REM Copy to shadow
  200 A%=screens  :REM set to file start
  210 Y%=-200:R%=640
  220 count%=0
  230 REPEAT
  240   PROCbackground
  250   CIRCLE FILL 640,Y%,R%
  260   buff%=USR(code+8):REM Write File
  270   A%=0
  280   CALL code+4 :REM Copy to shadow
  290   A%=buff%
  300   D%=1+R% DIV 50
  310   Y%+=D%+1:R%-=D%
  320   count%+=1
  330 UNTIL R%<=0 OR buff%=0
  340 IF buff%=0 THEN
  350   count%-=1:PRINT"RAM full"
  360 ELSE PRINT"RAM used ";buff%-screen
s
  370 ENDIF
  380 PRINTcount%;" Screens"
  390 REM-------------------------------
  400 REM          Display Routine
  410 VDU19,0,24,128,128,196:REM Border
  420 PRINTTAB(0,3);"Press space to Disp
lay"
  430 PRINT"Any other key to quit, and s
ave code"
  440 IF GET=32 THEN
  450   REPEAT
  460     PROCbackground
  470     CIRCLE FILL 640,-200,640
  480     Z=INKEY(100):A%=screens
  490     FOR N%=1 TO count%
  500       WAIT:WAIT
  510       IF A%>0 THEN A%=USR(code+12)
  520     NEXT
  530     Z=INKEY(300)
  540   UNTIL Z<>32 AND Z>-1
  550 ENDIF
  560 ON:PRINT"To save machine code, use
:"
  570 PRINT" *SAVE DeltaCode ";~code;" "
~P%
  580 END
  590 :
  600 DEFPROCbackground
  610 Z%=RND(-100)
  620 GCOL4+128:CLG:GCOL7
  630 FOR Z%=1 TO 30
  640   CIRCLE FILL RND(1280),RND(1024),
RND(4)
  650 NEXT
  660 GCOL128:GCOL2
  670 ENDPROC
  680 :--------------------------------
  690 DEFPROCassemble
  700 REM Call code to initialise
  710 REM Call code+4 to copy current
  720 REM screen to shadow screen
  730 :
  740 REM Call code+8 with A%=address
  750 REM creates next deltafile, and
  760 REM returns the updated address
  770 REM If zero is returned, the data
  780 REM is too long for the buffer
  790 :
  800 REM Call code+12 loads a deltafile
  810 REM back to the screen. If zero is
  820 REM returned, RAM is exceeded
  830 :
  840 scrnsize =1:REM Size of screen
  850 scrn1base=2:REM Base addr scrn1
  860 scrn2base=3:REM Base addr scrn2
  870 scrnpnt  =4:REM Ptr to scrn word
  880 scrn1word=5:REM Word from scrn1
  890 scrn2word=6:REM Word from scrn2
  900 point    =5:REM Ptr for display
  910 word     =6:REM Word for dsply
  920 fileaddr =7:REM Curr addr in file
```

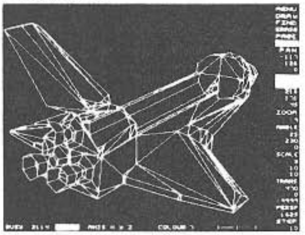
```
 930 temp     =8:REM Temp register
 940 endofbuff=9:REM End of buffer
 950 :
 960 FOR pass=0 TO 1
 970 P%=code
 980 [
 990 OPT pass*3
1000 B initialise
1010 B screencopy
1020 B screensave
1030 :
1040 .screenload
1050 STMFD R13!,{R14}
1060 BL getparams
1070 MOV fileaddr,R0
1080 MOV R0,#0
1090 LDR point,[fileaddr],#4
1100 CMP point,#&80000000
1110 BEQ scrnend
1120 .loadloop
1130 LDR word,[fileaddr],#4
1140 STR word,[scrn1base,point]
1150 LDR point,[fileaddr],#4
1160 CMP point,#&80000000
1170 BNE loadloop
1180 .scrnend
1190 MOV R0,fileaddr;First free addr
1200 LDMFD R13!,{PC}
1210 :
1220 .screensave
1230 STMFD R13!,{R14}
1240 BL getparams
1250 MOV fileaddr,R0
1260 MOV R0,#0
1270 MOV scrnpnt,#0
1280 .scrngetloop
1290 LDR scrn1word,[scrn1base,scrnpnt]
1300 LDR scrn2word,[scrn2base,scrnpnt]
1310 CMP scrn1word,scrn2word
1320 BEQ match
1330 STR scrnpnt,[fileaddr],#4
1340 STR scrn1word,[fileaddr],#4
1350 CMP fileaddr, endofbuff
1360 BHS saveend
1370 .match
1380 ADD scrnpnt,scrnpnt,#4
1390 CMP scrnpnt,scrnsize
1400 BLO scrngetloop
1410 MOV temp,#&80000000
1420 STR temp,[fileaddr],#4
1430 MOV R0,fileaddr ;1st free addr
1440 .saveend
1450 LDMFD R13!,{PC}
1460 :
1470 .screencopy
1480 STMFD R13!,{R14}
1490 BL getparams
1500 MOV R0,scrn2base
1510 .copyloop
1520 LDMIA (scrn1base)!,{R5-R12}
1530 STMIA (scrn2base)!,{R5-R12}
1540 CMP scrn1base,R0
1550 BLO copyloop
1560 LDMFD R13!,{PC}
1570 :
1580 .initialise
1590 ADR R0,data1
1600 ADR R1,data2
1610 SWI "OS_ReadVduVariables"
1620 LDR R2,[R1]
1630 LDR R3,[R1,#4]
1640 ADD R2,R2,R3
1650 STR R2,[R1,#8]
1660 MOV PC,R14
1670 :
1680 .getparams
1690 ADR temp,data2
1700 LDMIA (temp)!,{scrnsize,scrn1base,
scrn2base,endofbuff}
1710 MOV PC,R14
1720 :
1730 .data1
1740 EQUD 7
1750 EQUD 148
1760 EQUD -1
1770 :
1780 .data2
1790 EQUD 0 \size
1800 EQUD 0 \base
1810 EQUD 0 \base+size
1820 EQUD screens+bsize%-&100
1830 ]:NEXT
1840 ENDPROC
```

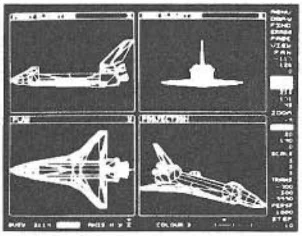
Next month we will make use of the Delta File generator listed here in a mouse-driven free-hand animator.

SILICON VISION

SOFTWARE FOR THE ARCHIMEDES & BBC

SolidCAD





[illegible] Architectural design. Interior design. Engineering [illegible] drawing in plan, front & side elevations and also directly [illegible] options for precision draughting and surface [illegible]. Also includes Sweep, Extrude & Macro [illegible] easily. Designs created with SolidCAD are [illegible] language for high speed flicker free animation [illegible] performs smooth shading for realism. [illegible] Realtime Solids Modeller (Arc) for £40.00

£49.95 (ARC & BBC) New

REALTIME SOLIDS MODELLER

[illegible] integrated design environment of SolidCAD and the [illegible] of a Realtime Graphics Language (RGL) module [illegible] supercharged performance. The package is ideal for [illegible] design, Engineering design & teaching CDT. The RGL [illegible] flicker free animation of designs from your own [illegible] performed for realistic images. Through our in house [illegible] techniques [illegible] the package can rival the design [illegible] Solids Modeller.

£89.95 (ARC) New

REALTIME GRAPHICS LANGUAGE

[illegible] a complete 3D Solids Wireframe [illegible] and 3D Editors for designing objects to animate [illegible] pixels [illegible] generator for fast 3D drawing [illegible] and Isometric [illegible] Also compatible with [illegible]

£49.95 (BBC)

SUPER-DUMP

[illegible] advantage of the highest resolution capability of [illegible] 1920 × 1024 resolution images can also be [illegible] printing. Fully compatible with SolidCAD [illegible] design system & 3D CAD Animation System [illegible] All packages can be made compatible with Super [illegible] commands. An example program is included in the [illegible]

£15.95 (BBC), £24.95 (ARC) New

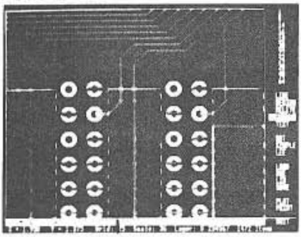
Presentation Manager

[illegible] create, edit and play back computer [illegible] demonstrations. Also handles graph plotting for [illegible] which can be incorporated within the [illegible]

£34.95 (BBC), £49.95 (ARC) New

SILICON VISION LTD. **SIGNAL HOUSE, LYON ROAD, HARROW**
MIDDLESEX HA1 2AG. TEL: 01-422 2274 or 01-861 2173
FAX: 01-427 5169. TELEX: 918266 SIGNAL G.
(Access/Mastercard/Eurocard accepted)
All prices include VAT and Carriage (Overseas orders add £4).

ARC-PCB



The ultimate PCB design system developed specifically for the Archimedes with a specification that cannot be matched. Includes Automatic routing. Rats-nesting. 8 layers. Surface mount capability. 0.001" resolution. 32" × 32" maximum board size. On-line Help. Fast Zoom Pan Redraw. Text & Silkscreen facility. Variable Line Pad Text Grid sizes. Part Libraries. Block Move Copy Rotate Mirror Erase options, and up to 300,000 components. For hardcopy, the system supports a large number of plotters and ordinary Epson compatible printers at very high resolutions (1920 × 1024) of 240 dots inch for near laser quality output. An entry level ARC PCB system (without auto routing) is available for £99.95 (Auto routing upgrade: £100). Enquire about our turnkey PCB design systems complete with Colour Archimedes and or plotters with prices from £1720 to £4905 [illegible]

£195.00 (ARC) New

RiscBASIC

Supercharge your Archimedes Basic programs by compiling them automatically into pure ARM Risc code with the RiscBASIC compiler. Features include Relocatable modules. Cross reference of all variables, functions and procedures. Floating point and Integer support. Stand alone code generator. Optimising compiler & Full [illegible] handler

£99.95 (ARC) New

RiscFORTH

A new 32 bit implementation of the FORTH 83 Standard, designed to take full advantage of the ARM architecture. Features include Multi tasking. Optimising compiler, built in ARM assembler with floating point [illegible] built in Full screen Editor. File system interface. OS calls support. Floating point & Integer maths. WIMP support. Single step debugger. Shadow screens for documentation. Block manipulation. Dictionary & [illegible]

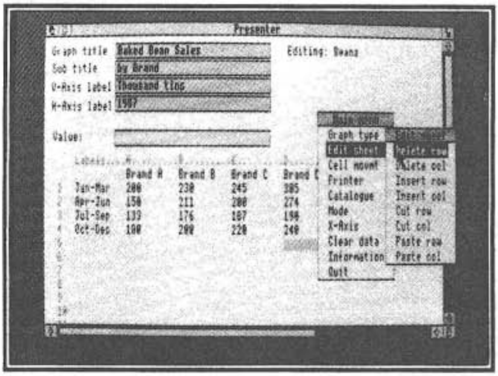
£99.95 (ARC) New

SEE US ON STAND 34 AT THE MICRO USER SHOW

PRESENTER

Mike Williams examines Presenter from Lingenuity, another contender in the business graphics stakes.

Last month in RISC User I reviewed Minerva's GammaPlot, a package for creating business graphics displays which I found both comprehensive and appealing in use. I was therefore interested to take a look at another package with similar aims. Presenter from Lingenuity, like GammaPlot, comes as a glossy package containing a 3.5" disc and manual, but here the slickness ends. The manual is a cheap looking and plain affair of just 21 printed pages. This fails to do justice to the content of the manual which explains most ideas simply, clearly and well.



Graph title: Baked Bean Sales
Sub title: by Brand
V-Axis label: Thousand tins
H-Axis label: 1987
Editing: Beans
Value:

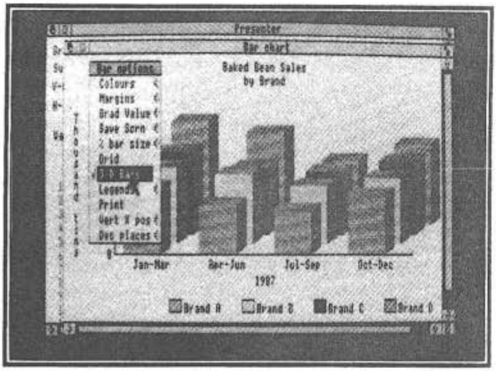
	Labels	A	B	C	D
		Brand A	Brand B	Brand C	Brand D
1	Jan-Mar	200	230	245	305
2	Apr-Jun	150	211	200	274
3	Jul-Sep	139	176	187	198
4	Oct-Dec	100	200	220	240
5					
6					
7					
8					
9					
10					

Data Entry Screen

A point to note about Presenter from the outset is that the software makes full use of the Archimedes WIMP system. The screen displays as a result look good, but scrolling, where necessary, can be painfully slow.

The initial screen display is the data entry screen (or window). This looks like a typical spreadsheet display with a potential for 25 rows by 9 columns of data, plus a legend for each row and another for each column, both user definable. Data may be entered into any cell from the keyboard, and the cursor can be set to move automatically to the next consecutive data entry position if required. Similarly the contents of any cell may be altered.

The data entry window also allows the user to specify a title and subtitle for the resulting graph, and labels for the X and Y axes. The spreadsheet layout offers more potential than that used by GammaPlot, which allows no more than two columns of data, but in many other respects I feel that GammaPlot offers both more features and greater flexibility.



Graph Display

Pressing the menu button on the mouse at any time brings up the main menu on the screen with a choice of 10 options, including 'Quit' to exit from the package. Once into Presenter, there appears to be no way, though, of entering any star command. This I feel is an unfortunate omission. The first and most immediately interesting option allows for a choice of graph type. Presenter offers just three: bar chart, line graph and pie chart. Once a bar chart has been drawn, a further option allows a choice of 2D or 3D bar chart. I find the range of choice disappointingly small - there are many other forms of data display which would be just as easy to implement, and which would add much to the variety and interest of the end result. There is also no facility for regression analysis or display of lines of best fit on scatter graphs.

Once a graph has been drawn (in a window), the mouse may be used to bring up a further menu (slightly different for each of the three types of graph display). This controls such features as the choice of colours (from a predefined set - Presenter can use mode 12 or mode 20), the spacing and labelling of graduations on the axes, the thickness of bars or lines, and the position of legends and axis labels. A graph can also be saved to disc, or printed (a print option in the main screen menu offers a choice of Epson compatible printers, the Integrex 132 colour printer or the Plotmate A4SM or A3M plotters).

There are no facilities for rescaling or resizing graphs, and thus no way of building up a display of several graphs together as is possible with GammaPlot. In general, Presenter's facilities for enhancing and embellishing graphs are quite meagre in comparison. However, Presenter is simple to use, and its displays are certainly clear and easy to read.

The main menu also provides an edit option with a number of functions which may be applied to the data entry sheets. Rows and columns may be inserted and deleted, and also copied and pasted back at alternative positions. A further important option is labelled 'Catalogue', and this gives a display of data files on disc. Files may be saved or loaded, and Presenter also has the ability to load in CSV files (comma separated values). This is a relatively standard format supported by many software packages running on a variety of machines. By this means, data from PipeDream, for example, may be output in CSV format, and then read into Presenter for display.

Presenter is certainly easy to use, but I am sure that it would have benefitted by offering a much wider choice of graph styles (horizontal bar charts, clustered or segmented bar charts for example). There is also less control over the detail of graphs compared with GammaPlot, but then Presenter does make everything seem very simple for the user, with features such as scaling and labelling of axes (just two examples) all happening quite automatically. GammaPlot certainly offers many more features for embellishing and customising graphs once drawn, but it could be argued that such aids are little more than cosmetic.

Overall, Presenter takes a simple, uncluttered approach to the task of displaying graphs, and this package may well appeal to many users at its comparatively low price. However, I feel that it could easily have achieved quite a lot more, and then presented a real challenge to Minerva's GammaPlot. Furthermore, I did not encounter the same sense of fun and excitement engendered by the latter, but that is something which you must decide for yourself.

***Presenter* (£29.84 inc VAT and p&p)**
is supplied by Lingenuity, P.O.Box 10, Halesworth, Suffolk IP19 0DX. Tel. (09868) 5476

COLOURING THE SUPER-CHARGED DISC MENU

Here, to round off David Pilling's Super-Charged Disc Menu, are some notes on the colour palette.

The version of the menu given last month contained instructions to give greater control over the colour palette. Here are some notes on how to define your own colours. Table 1 gives a breakdown of the colour numbers used for the different parts of the display. Also included are the line numbers on which these colours are defined.

Colour	Function	Line nos
0	Filename Text	2635/6
3	Banner Text	2651/2
4	Option Text	2653/4
5	Option Background	2655/6
7	Directories Text	2631/2
8	Files/Directories Bcgnd	2640/50
9	Main Background	2660/70
10	Banner Background	2680/90

Table 1 Menu Colour Definitions

To set up your own colours, all you have to do is to alter the definitions to give the required RGB components for each part. For example, to change colour 3 (the banner text colour) to yellow, use:

```
    2651 EQUB19:EQUB3:EQUB16:EQUB240
    2652 EQUB240:EQUB0
```

The last 3 numbers define the red green and blue components of the colour respectively.

Colour	RGB Components		
0	144	0	0
3	240	240	0
4	240	240	240
5	208	0	48
7	240	240	240
8	128	144	176
9	0	96	32
10	0	176	80

Table 2 Suggested New Palette

Table 2 gives a suggestion for an alternative set of colours. These give higher prominence to the main selection boxes, and a copy of this version of the menu is included on this month's magazine disc.

New
ARCHWAY

REVIEWS

ARCHEFFECT

Mike Williams has been trying out some new software on his Archimedes to good effect.

Wherever an Archimedes is on show you will often see the machine displaying a carousel of pictures. In many cases a variety of fancy effects will be used to change from one picture to the next, or to distort the picture on the screen. If you have watched these displays with admiration and maybe a tinge of envy then the answer is at hand in the form of Archeffect.

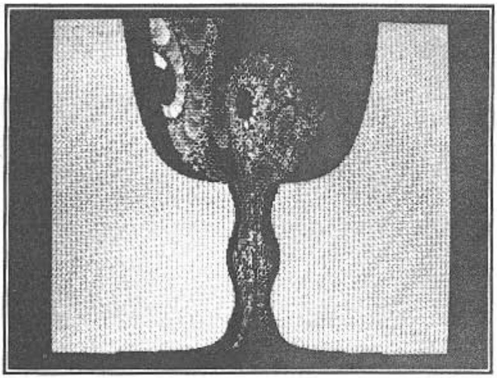
Archeffect, supplied on disc, is a relocatable module. Once this has been installed, star commands will allow you to set up your own impressive screen displays. The disc also contains a number of demonstrations and sample screen images for you to practice with. The software is accompanied by a well produced 40 page manual, though it sometimes lacks clarity and precision in its descriptions.

It would be pointless to attempt to describe all the commands in detail here, so I will concentrate on the main features. One characteristic that many commands have in common is that they rely on a screen image being stored somewhere in memory. Such a screen image could have been created and saved by another program, and four Mandelbrot displays are provided for you to use. Once a screen image has been loaded into memory, it can be transferred to the screen (in effect to screen memory) in a variety of ways.

Some of the star commands available control the way in which the image is 'put' on the screen, expanding outwards from the screen centre, rolling vertically down the screen or being pulled into place from one corner. Here, the final image is an accurate reproduction of that stored in memory.

Other star commands distort the image as it is transferred to the visible screen, by changing its size for example and positioning it wherever desired. Other forms of distortion allow the image to appear as though 'draped' over a solid sphere, or over the surface of a wine glass. Some of these commands can be used to good effect within loops so that a distorting image can seem to pass through a variety of intermediate shapes.

The module also has the facility to store up to five images in a data format within its own memory space. Each stored image can be distorted by a set of parameters, and the data saved to disc or reloaded from disc as required. Thus as well as the predefined distortions, the user can contrive some of his own.



FONTS

Archeffect also provides some star commands for using anti-aliased fonts, and a font called blocky is supplied on disc. You can also use Acorn's anti-aliased fonts supplied on the Archimedes Welcome disc. Archeffect's commands allow for a choice of font (and size), and for printing in the chosen font on the screen. Although useful in their own right, the provision of these commands is a little curious in the context.

IMAGE DESIGNER

The manual states that the purpose of this separate utility program is to design screen images, but its facilities are so limited in this respect that you would be well advised to use other means for this purpose. What the program does do is allow you to select any one of five internally saved designs, and then save this to disc for use by the various *IMAGE commands. These are the ones which allow a user defined distortion to be applied to a screen image (as mentioned already). The controlling menu for this also allows star commands to be entered so there is some useful purpose to be served.

THE MANUAL

Although clear and glossy, I found the manual at times confusing. This is particularly so with regard to the use of decimal or hexadecimal numbers in commands, and in the use of space or comma to separate parameters (commas alone cause errors). The use of a typeface with a rather peculiar rendering of the ampersand (&) character is also confusing.

I would like to have seen some more extended examples, particularly the use of the various star commands within loops which would better illustrate the potential of this software. Some suitable illustrations in such a highly graphics oriented package would not have come amiss either.

CONCLUSIONS

If you want to package up your screen displays with slick routines, then Archeffect certainly has a lot of potential, and it is comparatively cheap. Given the huge interest in graphics on the Arc, I would like to see this package developed and updated to keep in step with some of the highly stunning screen effects that have been seen recently on the Archimedes, and I hope the suppliers may still consider a mark II version. As it stands I would rate Archeffect good value at its price, but in the end the results will depend as much on your imagination and skill as they do upon this software.

[illegible]	[illegible]
[illegible]	[illegible]
	[illegible]
	[illegible]
	[illegible]
[illegible]	[illegible]

RISC USER TOOLBOX (5)

David Spencer adds a scrolling disassembler to the RISC User Toolbox.

An essential tool to any assembly language programmer is a disassembler to enable machine code programs to be examined. This month's addition to the RISC User Toolbox provides just that. As with the previous additions, the listing given here is just a set of lines to add to the existing program from the first four parts. Before adding the new lines make sure that the existing program has not been renumbered. Special care is needed with this listing because many of the lines either replace existing ones or slot between them.

As many of the changes are quite subtle, it is probably better to type in the new lines, save them as a spooled file (using *SPOOL), and then load in the original program and append the new lines with *EXEC. See the User Guide for more details of *SPOOL and *EXEC. Once all the new lines are added save the program under a different name, and run it to assemble the new Toolbox module. The assembled module is loaded as in previous months. The disassembler uses a SWI call contained within the Debugger module, and therefore this must be present for it to work properly.

The disassembler can be called up in two ways. First, you can use *MEDIT to start the memory editor and then switch to the disassembler, or secondly, you can start the disassembler directly using:

```
*DISASS <address>
```

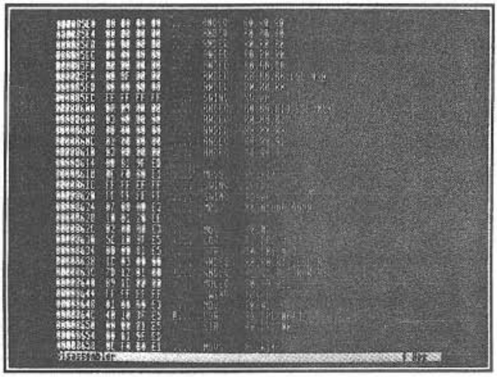
where <address> is the address in hex at which disassembly will start. For example:

```
*DISASS 8F00
```

Both these methods are essentially the same, as once started you can flick between the memory editor and disassembler displays by pressing the Insert key.

From within the disassembler, memory can still be altered, and the same controls apply as those used for the memory editor. You will also notice that the different areas of both memory editor and disassembler displays are coloured. This makes them easier to read.

Another feature of the disassembler is the ability to display floating point (and other co-processor) instructions in a different colour, or to treat them as illegal instructions. The current state is shown by the letter 'f' in the status line. A lowercase letter means treat floating point instructions as illegal, while upper case means display them, but in yellow rather than green. You can change between the two settings using function key F12.



Next month we will add further features to the RISC User Toolbox. In the mean time, if you have any suggestions for facilities to include, please drop us a line.

```
 332 EQUS "Disass":EQUB 0
 333 ALIGN:EQUD disassc:EQUD &10001
 334 EQUD dissyn:EQUD dishlp
1403 .dishlp EQUS "*Disass invokes the
memory editor at the given address with
a disassembler display.":EQUB 13
1404 .dissyn EQUS "Syntax: Disass <addr
ess>":EQUB 0:ALIGN
2430 BL valadd:BL swi10
2450 BL frange:ADR R5,mtxt
2658 EQUS "Disassemble":EQUB 0
2659 EQUS "DisassLine":EQUB 0
2660 EQUS "SetMode":EQUB 0
3714 B swi8:B swi9:B swi10
4801 STMFD R13!,{R6}:LDR R6,[R12,#28]
4802 CMP R6,#0:MOVEQ R6,#15:MOVNE R6,#3
```

```
 4810 SUB R0,R0,#1:AND R5,R0,R6
 4820 CMP R5,R6:BEQ curup2
 4829 LDMFD R13!,{R6}
 4861 STMFD R13!,{R6}:LDR R6,[R12,#28]
 4862 CMP R6,#0:MOVEQ R6,#15:MOVNE R6,#3
 4870 AND R5,R0,R6:CMP R5,#0
 4880 BEQ curdown2:LDMFD R13!,{R6}:MOV P
C,R14
 4891 STMFD R13!,{R6}:LDR R6,[R12,#28]
 4892 CMP R6,#0:MOVEQ R6,#16:MOVNE R6,#4
 4900 ADD R0,R0,R6:CMP R0,R3

 4910 SUBCC R6,R6,#1:BCC curdown2:SUB R0
,R0,R6
 4919 LDMFD R13!,{R6}
 4940 STMFD R13!,{R0-R1,R14}
 4955 ADD R6,R6,#1:MOV R1,#15:MUL R6,R1,
R6
 4960 ADD R0,R0,R6:CMP R0,R3
 5020 LDMFD R13!,{R0-R1,R14}:LDMFD R13!,
{R6}:MOV PC,R14
 5030 .curup STMFD R13!,{R6}:LDR R6,[R12
,#28]
 5031 CMP R6,#0:MOVEQ R6,#16:MOVNE R6,#4
 5032 CMP R0,R6:LDMCCFD R13!,{R6}:MOVCC
PC,R14
 5040 SUB R0,R0,R6:CMP R0,R2:SUBCS R6,R6
,#1:BCS curup2
 5050 ADD R0,R0,R6:LDMFD R13!,{R6}:MOV P
C,R14
 5060 .curup2:STMFD R13!,{R0-R1,R14}
 5075 ADD R6,R6,#1:MOV R1,#15:MUL R6,R1,
R6
 5080 SUBS R0,R0,R6:BMI curup25
 5150 LDMFD R13!,{R0-R1,R14}
 5151 LDMFD R13!,{R6}:MOV PC,R14
 5190 .prtlinex STMFD R13!,{R0-R1,R14}
 5191 LDR R1,[R12,#28]:CMP R1,#0:LDR R1,
[R13,#4]
 5192 BICNE R0,R0,#3:BLNE disass
 5193 LDMNEFD R13!,{R0-R1,PC}
 5210 LDMFD R13!,{R0-R1,PC}
 5221 LDR R1,[R12,#32]
 5250 ADD R1,R12,#64:MOV R2,#10
 5310 .addone SWI &111:SWI &108:SWI "OS_
Write0"
 5332 SWI &111:SWI &109
 5340 .prtloop:ADD R1,R12,#64
 5392 SWI &111:SWI &10A
 5490 .prtscr STMFD R13!,{R0-R5,R14}
```

```
 5491 LDR R5,[R12,#28]:CMP R5,#0
 5492 MOVEQ R5,#16:MOVNE R5,#4
 5500 SWI &11E:SUB R4,R5,#1:BIC R0,R0,R4
 5505 MOV R4,#15:MUL R4,R5,R4
 5510 SUB R0,R0,R4:MOV R4,#31
 5590 .prtscr3 CMP R5,#4:BLEQ disass:BEQ
prtscr4:BL prtline
 5610 BEQ prtscr5:ADDS R0,R0,R5
 5670 LDMEQFD R13!,{R0-R5,PC}^
 5711 LDR R2,[R12,#28]:CMP R2,#0
 5712 MOVEQ R2,#15:MOVNE R2,#3
 5720 SWI &11F:AND R0,R0,R2:CMP R1,#0
 5735 ADR R3,offtab:LDR R2,[R12,#28]
 5736 ADD R3,R3,R2,LSR #3:ADD R3,R3,R1
 5740 LDRB R3,[R3]:ADD R0,R0,R3
 5761 SWI &111:CMP R3,#0:SWIEQ &10A:SWIN
E &109
 5841 .offtab EQUB 11:EQUB 60
 5842 EQUB 10:EQUB 23
 5881 STR R1,[R12,#32]
 5896 AND R6,R1,#16:STR R6,[R12,#28]
 5897 AND R6,R1,#8:STR R6,[R12,#36]
 5981 MOV R0,#221:MOV R1,#&C0
 5982 BL byte0:STRB R1,[R12,#9]
 6071 CMP R5,#&CD:LDREQ R5,[R12,#28]
 6072 EOREQ R5,R5,#16:STREQ R5,[R12,#28]
 6073 BLEQ prtscr:BEQ keydone
 6074 CMP R5,#&CC:LDREQ R5,[R12,#36]
 6075 EOREQ R5,R5,#8:STREQ R5,[R12,#36]
 6076 BLEQ prtscr:BEQ keydone
 6170 BNE c5:LDR R5,[R12,#28]
 6171 CMP R5,#0:BICEQ R0,R0,#15
 6172 BICNE R0,R0,#3:B keydone
 6180 .c5 CMP R5,#&8D:BNE c6
 6181 LDR R5,[R12,#28]:CMP R5,#0
 6182 ORREQ R0,R0,#15:ORRNE R0,R0,#3
 6183 B keydone:.c6 CMP R5,#&8E
 6190 BNE notdn:LDR R5,[R12,#28]
 6191 CMP R5,#0:ADDEQ R5,R0,#16*31
 6200 ADDNE R5,R0,#4*31
 6240 LDR R5,[R12,#28]:CMP R5,#0
 6241 MOVEQ R5,#16*31:MOVNE R5,#4*31
 6242 SUBS R5,R0,R5:BCC keydone
 6541 MOV R0,#221:LDRB R1,[R12,#9]
 6542 BL byte0
 6550 LDR R1,[R12,#32]:AND R1,R1,#1
 6551 LDR R4,[R12]:ORR R1,R1,R4
 6552 LDR R4,[R12,#24]:ORR R1,R1,R4,LSL
#2
 6553 LDR R4,[R12,#36]:ORR R1,R1,R4
 6554 LDR R4,[R12,#28]:ORR R1,R1,R4
 6555 SWI &111:SWI &107
 6650 ADD R1,R1,#1:CMP R1,#68:BNE st1
 6680 CMP R1,#68:BNE st2
 6690 .st3 LDR R0,[R12,#28]:CMP R0,#0
 6691 BNE st4:SWI &120:B st5
 6692 .st4 LDR R0,[R12,#36]:CMP R0,#0
 6693 MOV R0,#ASC"F":ORREQ R0,R0,#&20
 6694 SWI "OS_WriteC"
 6695 .st5 SWI &120:LDR R0,[R12]:CMP R0,
#0
 7350 BL swi10:ORR R1,R1,#1
 7700 BL swi10:ORR R1,R1,#1
 9610 BL swi10
10600 .swi10 STMFD R13!,{R0-R2,R14}
10610 SWI &116:SWI &10C:ADR R1,coldat
10620 .swi102 LDR R2,[R1],#4:CMP R2,#0
10630 LDMEQFD R13!,{R0-R2,PC}
10640 SWI &113:AND R0,R2,#&FF
10650 SWI "OS_WriteC":SWI &110
10660 MOV R2,R2,LSR #8:AND R0,R2,#&FF
10670 SWI "OS_WriteC":MOV R2,R2,LSR #8
10680 AND R0,R2,#&FF:SWI "OS_WriteC"
10690 MOV R0,R2,LSR #8:SWI "OS_WriteC"
10700 B swi102
10710 .coldat EQUB 8
10720 EQUB 0:EQUB 240:EQUB 240
10730 EQUB 9
10740 EQUB 240:EQUB 240:EQUB 240
10750 EQUB 10
10760 EQUB 240:EQUB 240:EQUB 0
10770 EQUB 11
10780 EQUB 16:EQUB 240:EQUB 64
10790 EQUB 12
10800 EQUB 240:EQUB 0:EQUB 240
10810 EQUB 13
10820 EQUB 240:EQUB 64:EQUB 0
10830 EQUB 14
10840 EQUB 220:EQUB 220:EQUB 220
10850 EQUB 15
10860 EQUB 220:EQUB 220:EQUB 220
10870 EQUD 0
10880 .swi8 STMFD R13!,{R0,R3-R6,R14}
10890 MOV R0,#18:MOV R6,R1:ADR R1,debug
10900 SWI "XOS_Module":SUB R5,R3,R6
10910 ADRVS R1,undef:LDRVS R2,undl:LDMVS
FD R13!,{R0,R3-R6,PC}:LDR R0,[R13]
10920 SWI "Debugger_Disassemble"
10930 STMFD R13!,{R1-R2}
10940 .swi82 LDRB R4,[R1],#1
```

```
10950 CMP R4,#0:LDMEQFD R13!,{R1-R2}
10960 LDMEQFD R13!,{R0,R3-R6,PC}^
10970 CMP R4,#ASC"#":ADDEQ R1,R1,#1
10980 CMP R4,#ASC"&":BNE swi82
10990 MOV R3,R1:MOV R0,#16
11000 SWI "OS_ReadUnsigned"
11010 LDR R0,[R13,#8]:SUB R0,R2,R5
11020 BIC R0,R0,#&FC000000:MOV R1,R3
11030 LDRB R4,[R1,#8]:MOV R2,#10
11040 SWI "OS_ConvertHex8"
11050 STRB R4,[R1]:B swi82
11060 .debug EQUS "Debugger":EQUB 0
11070 .swi9 STMFD R13!,{R0-R1,R3-R7,R14}
11080 STMFD R13!,{R2}
11090 ADD R3,R12,#128:MOV R4,#&11
11100 STRB R4,[R3],#1:MOV R4,#8
11110 STRB R4,[R3],#1:MOV R4,#32
11120 STRB R4,[R3]:STRB R4,[R3,#1]
11130 STRB R4,[R3,#2]:STRB R4,[R3,#3]
11140 LDMFD R13,{R2}:ADD R0,R0,R2
11150 TST R1,#1:MOV R1,R3
11160 MOV R2,#10:BNE swi92
11170 ADD R1,R1,#4:SWI "OS_ConvertHex4"
11180 B swi93
11190 .swi92 SWI "OS_ConvertHex8"
11200 .swi93 ADD R3,R3,#8
11210 STRB R4,[R3],#1:STRB R4,[R3],#1
11220 MOV R4,#&11:STRB R4,[R3],#1
11230 MOV R4,#9:STRB R4,[R3],#1
11240 MOV R5,#4:LDR R6,[R13,#4]
11250 .swi94 LDRB R0,[R6],#1
11260 MOV R1,R3:MOV R2,#10
11270 SWI "OS_ConvertHex2":MOV R4,#32
11280 STRB R4,[R3,#2]:ADD R3,R3,#3
11290 SUBS R5,R5,#1:BNE swi94
11300 STRB R4,[R3],#1
11310 MOV R4,#&11:STRB R4,[R3],#1
11320 MOV R4,#10:STRB R4,[R3],#1
11330 MOV R5,#4:LDR R6,[R13,#4]
11340 .swi95 LDRB R0,[R6],#1
11350 CMP R0,#32:MOVCC R0,#ASC"."
11360 CMP R0,#127:MOVCS R0,#ASC"."
11370 STRB R0,[R3],#1:SUBS R5,R5,#1
11380 BNE swi95:MOV R4,#32
11390 STRB R4,[R3],#1:STRB R4,[R3],#1
11400 LDMFD R13!,{R1}:LDR R0,[R13]
11410 ADD R1,R0,R1:LDR R0,[R0]:BL swi8
11420 MOV R4,#&11:STRB R4,[R3],#1
11430 LDMFD R13,{R0}:LDRB R0,[R0,#3]
11440 AND R0,R0,#15:CMP R0,#12:BCC swi96
```

```
11450 CMP R0,#15:BCS swi96:LDR R5,[R13,#4]
11460 TST R5,#8:MOVNE R4,#12:BNE swi97
11470 ADR R1,undef:LDR R2,undl
11480 .swi96 LDR R4,[R1]:ADR R5,undef
11490 LDR R5,[R5]:CMP R4,R5
11500 MOVEQ R4,#13:MOVNE R4,#11
11510 .swi97 STRB R4,[R3],#1
11520 .swi98 LDRB R4,[R1],#1
11530 STRB R4,[R3],#1:SUBS R2,R2,#1
11540 BNE swi98:MOV R4,#&11
11550 STRB R4,[R3],#1:MOV R4,#7
11560 STRB R4,[R3],#1:MOV R4,#0
11570 STRB R4,[R3],#1:ADD R2,R12,#128
11580 LDMFD R13!,{R0-R1,R3-R7,PC}^
11590 .undef EQUS "Undefined instruction":EQUB 0
11600 .undefe ALIGN
11610 .undl EQUD undefe-undef
11620 .disass STMFD R13!,{R0-R2,R14}
11630 LDR R1,[R12,#32]:BIC R1,R1,#8
11640 LDR R2,[R12,#36]:ORR R1,R1,R2
11650 LDR R2,[R12,#12]
11660 BL swi9:.disass2 LDRB R0,[R2],#1
11670 CMP R0,#0:BEQ disass1:SWI "OS_WriteC"
11680 B disass2:.disass1 MOV R0,#134
11690 SWI "OS_Byte":RSB R1,R1,#80
11700 .disass3 SWI &120:SUBS R1,R1,#1
11710 BNE disass3:LDMFD R13!,{R0-R2,PC}^
11720 .disassc
11730 LDR R12,[R12]
11740 STMFD R13!,{R14}:MOV R1,R0
11750 MOV R0,#16:SWI "OS_ReadUnsigned"
11760 MOV R0,R2:MOV R1,R2
11770 BL valadd:BL swi10
11780 BL frange:ADR R5,dtxt
11790 MOV R1,#17:MOV R4,#0
11800 .disassc2
11810 STMFD R13!,{R2-R5}:BL swi0
11820 MOV R6,R2:LDMFD R13!,{R2-R5}
11830 CMP R6,#27:SWI &11F
11840 SWI &100:SWI &11F
11850 BNE disassc2:SWI &10A:LDMFD R13!,{PC}
11860 .dtxt
11870 EQUS "Disassembler"
11880 EQUB 0
```

A SIMPLE CUSTOMISED INPUT FUNCTION

by Lee Calcraft

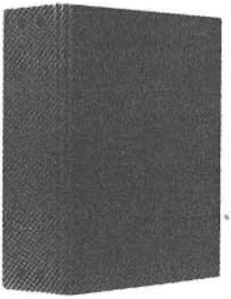
The simple function listed here gives the user a little more flexibility than is offered by Basic's INPUT function. Its chief merit is that it allows you to specify how many characters may be input. Basic's INPUT function always allows up to 238 characters. This is often inconvenient, since it means that input handling cannot be foolproof, and if someone leaves a key pressed down, text will be printed all over the place.

The function FNinput allows the programmer to specify the maximum length of string which will be accepted. Entries longer than this cause a beep. You can also specify the range of ASCII characters that the routine will accept, although there is a snag with this. The SYS call used by the function chooses to echo all characters input, even those out of range, with the result that there is no restriction on the number of these characters written to the screen, and the main advantage of the routine is lost.

As you can see, the function FNinput has four parameters. The first two are the lowest and highest ASCII characters accepted for input. The third is the maximum number of characters allowed, and the fourth is a flag specifying whether the input should be treated as numerical or string. It should be set to TRUE for a string. If you are using the function in your own program, you will also need to reserve a small area of RAM as a text buffer, as we have done in line 60.

The accompanying program demonstrates how the routine works. It requests two inputs - a string and then a numerical value - and then prints them out. The maximum lengths are set at 20 and 6 respectively, and you can easily check the effect of exceeding these. When you are typing in the routine, be careful with the commas in the OS_ReadLine parameters, especially the ones before buff% and len%. Incidentally on exit from the function the variable len% is automatically set to the length of string supplied by the user.

```
   10 REM >UsrInput
   20 REM Customised INPUT using
   30 REM SYS OS_ReadLine
   40 REM by Lee Calcraft
   50 :
   60 DIM buff% &100:REM Reserve RAM
   70 MODE12
   80 PRINT"Customised INPUT Function"'
   90 PRINT"String please ";
  100 string$=FNinput(32,126,20,TRUE)
  110 PRINT'string$
  120 :
  130 PRINT''"Now a number ";
  140 value=FNinput(32,126,6,FALSE)
  150 PRINT'value
  160 END
  170 :
  180 DEFFNinput(lochr%,hichr%,maxlen%,t
ext)
  190 SYS "OS_ReadLine",buff%,maxlen%,lo
chr%,hichr% TO ,len%
  200 IF text:=$buff% ELSE =VAL($buff%)
```



INTRODUCING ARM ASSEMBLER (7)

This Month Lee Calcraft investigates the Barrel Shifter, and Shift and Rotate Operations.

THE BARREL SHIFTER

The Barrel Shifter is a special piece of hardware within the ARM processor chip for performing shift and rotation operations. It is separate from the Arithmetic-Logic Unit (or ALU), which performs the CPU's arithmetical and logical operations. This means that the barrel shifter can be used without detracting from the speed of the ALU. In fact, Acorn has placed the barrel shifter in the path of one of the data inputs to the CPU. In consequence, all of the ARM's logical and arithmetic instructions have an option to shift or rotate the right-hand operand.

Taking the ADD instruction as an example, if we wish to add the contents of R2 to R1, and place the result into R0, we could use:

```
    ADD R0,R1,R2
```

But we could equally well shift the contents of R2 by say 16 bits to the left before the addition takes place using:

```
    ADD R0,R1,R2,LSL #16
```

It should be stressed that the shifted contents of R2 do not get written back to that register. The shift takes place as the data stream passes through the barrel shifter on its way to the ALU.

Since a binary shift to the left by n places multiplies the operand by 2^n, and a similar shift to the right performs an integer division by the same amount, it is easy to see how useful the ARM's multiple shift and rotate operations can be. Remember that on "coal-fired" CPUs like the Z80 and 6502, shift and rotation operations work only one bit at a time, and each requires the full time of the CPU. On the ARM by contrast, shifts and rotations performed in this way may be from 0 to 31 bits in magnitude, and providing that the shift is expressed as an immediate operand, there is no time overhead whatsoever; though if the degree of shift is given in a register this doubles the execution time. Thus the ADD instruction above would take just 125 nano-seconds to perform in RAM on an Archimedes. Adding condition and flag-setting suffixes to the instruction also carries no time overhead. So for example, the instruction:

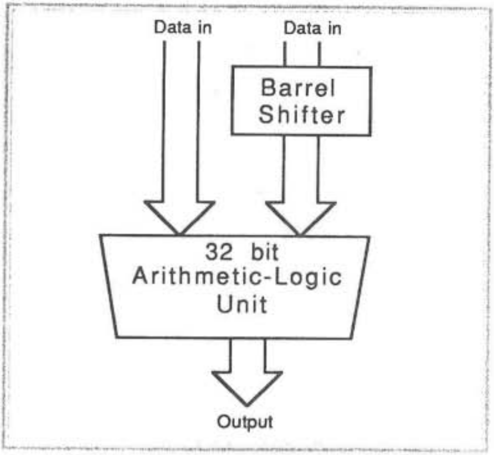


Fig 1 The barrel shifter is situated in the path of one of the inputs to the ALU

```
    SUBNES R0,R1,R2,ROR #8
```

still takes only 125 nano-seconds.

SPECIFYING A SHIFT

There are four possible shift mnemonics: LSL, ASL, LSR and ASR (see table 1). The first two are identical in effect. They cause a left shift of the binary operand. With a shift of n bits to the left using LSL #n, n zero bits are shifted in at the right of the operand, and the carry flag holds the last bit to be shifted out at the far left.

Thus the number:

```
  C? 11001100 11001100 11001100 11001100
```

becomes:

```
  C1 10011001 10011001 10011001 10011000
```

after LSL #1, where "C" indicates the carry flag.

The effect of LSR is very similar, except that zeros are added at the left-hand end, and the carry flag holds the bit last shifted out at the right-hand end. After LSR #1, our number would become:

```
C0 01100110 01100110 01100110 01100110
```

The effect of ASR is somewhat different. This performs a shift to the right in a similar way to LSR, but to assist with signed integer arithmetic, the sign bit (bit 31) of the original operand is preserved. Thus instead of introducing zeros at the left-hand side for each bit-shift, a copy of bit 31 is introduced. Using the two's complement sign convention, this would mean that with positive integers, a zero would be introduced, while with negative ones, a one would be introduced. After ASR #1 our number would become:

```
C0 11100110 01100110 01100110 01100110
```

If we had used ASR #2, the result would have been:

```
C0 11110011 00110011 00110011 00110011
```

Thus when using ASR #n on a negative number, n ones are introduced at the left; while with a positive number, n zeros are introduced.

THE SHIFT MNEMONICS	
LSL	Logical shift left
ASL	Arithmetic shift left
LSR	Logical shift right
ASR	Arithmetic shift right
THE ROTATE MNEMONICS	
ROR	Rotate right
RRX	Rotate right with extend

Table 1. Shift and Rotate

SPECIFYING ROTATION

There are just two rotation mnemonics, ROR (Rotate Right), and RRX (Rotate Right with eXtend), and only one of these, the former, takes a parameter. ROR performs a standard rotate. The reason that there is no left-rotating counterpart is that rotating right by 8 bits is the same as rotating left by 24 bits. After ROR #1 all bits are shifted to the right by one position, and the bit lost at the right-hand end appears as bit 31 of the new number. The carry flag also takes the value of the last bit shifted out. After such a rotation, the number used in our examples above would become:

```
C0 01100110 01100110 01100110 01100110
```

If it were allowed, a rotate right by 32 bits would thus leave any register unchanged. But since this is a useless exercise, the instruction code which would have performed ROR #32 has been allocated to the RRX instruction.

Rotate right with extend performs exactly the same operation as the 6502's ROR. It has no parameter, and can only rotate by one bit at a time. It differs from ROR #1 in that the carry flag is treated as bit 32. Thus with RRX, the contents of the carry flag are rotated into bit 31, and at the other end of the register, the contents of bit zero are rotated into the carry flag. This forms a 33 bit loop. If we perform an RRX on the following number:

```
C1 11001100 11001100 11001100 11001100
```

the result will be:

```
C0 11100110 01100110 01100110 01100110
```

Oddly enough, there is no left-handed equivalent of RRX. But you can achieve exactly the same result by using the following ADC instruction:

```
ADCS R0,R0,R0
```

SPECIFYING THE DEGREE OF SHIFT

With each of the shift and rotate operations which take a parameter, the degree of shift may be specified as an immediate value between 0 and 31, or it may be a register. In the latter case, only the low byte of the register's contents will be used. Here is an example of the two forms:

```
ANDEQ  R0,R1,R2,LSL #4
ADCLOS R0,R2,R2,ROR R10
```

Both instructions are conditional, and in addition, the second sets flags to reflect the result. The first shifts the contents of R2 by 4 places to the left (thus multiplying it by 16), before adding it to the contents of R1, and placing the result in R0. The second rotates the contents of R2 by an amount specified in the low byte of register R10, before adding the result to the original contents of R2, and placing the final result in R0.

It should again be stressed that in both of these examples, the contents of R2 remain unchanged after the operation. But there is one way in which the result of the shift can leave its mark directly. On all logical instructions from the arithmetic and logical set, if the S suffix is specified, then the carry flag reflects the result of any shift or rotate operation carried out on the second operand. By contrast, the N and Z flags reflect the result of the complete operation. Thus after the following instruction:

```
    ORRS R0,R1,R2,LSL #1
```

N and Z will be set according to the overall result of the OR operation, while the carry flag will hold the top bit of the contents of R2 (since LSL #1 shifts the top bit of a number into the carry flag).

PUTTING IT TO USE

We can now take a brief look at some of the ways in which the power of the barrel shifter can be harnessed. First of all, how can it be used to perform a simple shift or rotation operation on a specified register? The answer is: by using the MOV instruction. To shift the contents of R0 to the left by 8 places, we can use:

```
    MOV R0,R0,LSL #8
```

If you want the carry flag to reflect the result of the shift, add the S suffix, thus:

```
    MOVS R0,R0,LSL #8
```

By using left or right shifts we can multiply or divide numbers by a variety of factors. The example above multiplies the contents of R0 by 256. Generally speaking we can multiply by 2^n, 2^n+1 or 2^n-1 in a single operation. 2^n is achieved using MOV as above. For 2^n+1 we can use the ADD instruction:

```
    ADD R0,R1,R1,LSL #n
```

This shifts the contents of R1 by n bits and adds the result to the contents of R1 (giving a multiplier of 2^n+1). To multiply by 2^n-1 we need to use the Reverse Subtract instruction, which we have not yet covered in any detail:

```
    RSB R0,R1,R1,LSL #n
```

will multiply the contents of R1 by 2^n-1.

Multiplying or dividing by other fixed factors can be achieved with combinations of instructions. For example the following pair of instructions will multiply the contents of R0 by 10, and place the result back into R0. Note the economy of register use in this example:

```
    MOV R0,R0,R0,LSL #1
    ADD R0,R0,R0,LSL #2
```

The first instruction doubles the contents of R0, and the second multiplies it by 5, giving the required result of 10 times.

By this point you may be wondering why we do not use the perfectly adequate 32-bit multiply instructions in the ARM's repertoire. The answer is that they take considerably longer to execute. A 32-bit multiply can take up to 2125 nano-seconds (depending on the exact values of the operands). If we can achieve the same result in one or two 125 nano-second instructions using selected shifts, then we obtain a speed increase of some 8 to 17 times. But I don't want to leave the impression that the only use for shift or rotation operations is for faster division or multiplication. There are many instances where such operations are used to extract selected parts of a 32 bit word, and for bit and flag manipulation in general. For example, the following instruction will place the top byte of the contents of R1 into the bottom byte of R0:

```
    MOV R0,R1,LSR #24
```

Space constraints prevent us from giving further examples. Even so, you will probably have gathered by now that the ARM's barrel shifter is an exceedingly powerful tool, the more so because it carries no time overhead in most circumstances. The only problem for the programmer is how to make the best use of the flexibility which it offers.

Next month we will take a closer look at the ARM's logical and arithmetic instruction set

HINTS & TIPS HINTS & TIPS

Compiled by Lee Calcraft.

PRINTER BUFFER TEST DEBUGGED

ADVAL(-4) can be used to test whether a printer is on line or not, since it returns the current free space in the Arc's printer buffer. The way to perform the test is to obtain the number of free bytes in the buffer, then place a few null characters into the buffer, and test the length again. If the number of free bytes has not changed, the printer must be connected, since it has absorbed the extra characters.

But there is a snag. The Arc is so fast that if you perform the test normally, you will always get the impression that there is no printer connected. The way around this is to incorporate a delay so that the printer can catch up. Here is the modified routine, presented as a function. It returns TRUE if a printer is connected, and FALSE if not. As a bonus, the accompanying program also prints out the size of the buffer in use. This will be 63 bytes (1023 bytes on RISC OS) unless you are using an extended buffer, such as that published in RISC User Volume 1 Issue 3.

```
   10 REM >PrintTest
   20 REM Arc Printer Test
   30 REM by Lee Calcraft
   40 :
   50 PRINT'"** PRINTER TEST **"'
   60 IF FNprinter THEN
   70 PRINT"Printer on line"
   80 ELSE
   90 PRINT"No printer detected":VDU7
  100 ENDIF
  110 PRINT'"Buffer size= ";startsize;"
bytes"'
  120 END
  130 :
  140 DEFFNprinter
  150 startsize=ADVAL(-4)
  160 VDU2,1,0,1,0,1,0,1,0,1,0,3
  170 wait=INKEY(2)
  180 endsize=ADVAL(-4)
  190 =(startsize=endsize)
```

USING DEF REMS

If you use strings of dotted or dashed lines to divide up long program listings into sections, you will be pleased to hear that if you place these outside of the main program loop, and outside of procedure and function definitions, they carry virtually no time penalty, because the Basic interpreter only comes across them when it is searching for the definition of a new procedure or function. This also means that you do not need to begin the line with a REM - the line can begin with the dots or dashes themselves.

Better still, if you start such lines with the keyword DEF, you cause no great problem to the interpreter. But now, if you wish to get a summary of a large program by turning on the printer, and tying:

```
   LIST IFDEF
```

The lines at which all procedures and functions are defined will be listed, together with all your comment lines beginning with DEF. This helps clarify a long program listing. For example, you might use something like:

```
DEF------------------------------
DEF-        Menu Choices
DEF------------------------------
```

put menu choice procedures here

```
DEF------------------------------
DEF-      Graphics Routines
DEF------------------------------
```

put graphics routines here,

etc.

The reason for the dash after the DEF is to save the Basic interpreter looking along the blank spaces when it is checking for a new procedure or function definition.

DETERMINING FILE LENGTH

A simple way to determine the length of a file without using SWI calls is to use EXT#. Simply open the file, and read EXT#. Thus:

```
   handle%=OPENIN (name$)
   PRINT EXT# handle%
   CLOSE# handle%
```

FASTER BASIC

As you may know, the ARM processor runs faster when accessing RAM than ROM. This is because the ROMs used in the machine are not guaranteed to work as fast as the machine's RAM chips. This is why you can speed up Basic by using RAM Basic. But there is another way. If you execute the following:

```
   SYS "OS_UpdateMEMC",64,64
```

you will instruct the Arc to access ROM at RAM speed. In many cases everything will work ok, and you will get a 20% speed increase. If not, no harm is done, just switch off or use Ctrl-Reset to reset your machine. An alternative way to reset the speed is to use:

```
   SYS "OS_UpdateMEMC",0,64
```

Thanks to Barry Christie for this.

ERROR 37

Basic error number 37 appears not to be documented in the *User Guide*. Its associated message is *No room for function/procedure call at line n*, and it occurs when the Basic stack runs out of room to stack the return addresses of nested procedures or functions. Since the stack normally has room for many thousands of nested return addresses, the error is only likely to occur if a procedure repeatedly calls itself in an infinite recursion. For example:

```
   10 PROCcrash
   20 END
   30 :
   40 DEFPROCcrash
   50 PROCcrash
   60 ENDPROC
```

INVERTING FLAGS

Flags which may either be TRUE (=-1) or FALSE (=0) may be inverted with the expression:

```
   flag=NOT flag
```

RISC USER magazine

MEMBERSHIP

RISC User is available only on subscription at the rates shown below. Full subscribers to RISC User may also take out a reduced rate subscription to BEEBUG (the magazine for the BBC micro and Master series).

All subscriptions, including overseas, should be in pounds sterling. We will also accept payment by Connect, Access and Visa, and official UK orders are welcome.

RISC USER SUBSCRIPTION RATES		RISC USER & BEEBUG
£14.50	1 year (10 issues) UK, BFPO, Ch.I	£23.00
£20.00	Rest of Europe & Eire	£33.00
£25.00	Middle East	£40.00
£27.00	Americas & Africa	£44.00
£29.00	Elsewhere	£48.00

BACK ISSUES

We intend to maintain stocks of back issues New subscribers can therefore obtain earlier copies to provide a complete set from Vol.1 Issue 1. Back issues cost £1.20 each. You should also include postage as shown:

Destination	UK, BFPO, Ch.Is	Europe plus Eire	Elsewhere
First Issue	40p	75p	£2
Each subsequent Issue	20p	45p	85p

MAGAZINE DISC

The programs from each issue of RISC User are available on a monthly 3.5" disc. This will be available to order, or you may take out a subscription to ensure that the disc arrives at the same time as the magazine. The first issue (with six programs and animated graphics demo) is at the special low price of £3.75. The disc for each issue contains all the programs from the magazine, together with a number of additional items by way of demonstration, all at the standard rate of £4.75.

MAGAZINE DISC PRICES	UK	Overseas
Single issue discs	£ 4.75	£ 4.75
Six months subscription	£25.50	£30.00
Twelve months subscription	£50.00	£56.00

Disc subscriptions include postage, but you should add 50p per disc for individual orders.

All orders, subscriptions and other correspondence should be addressed to:

RISC User, Dolphin Place, Holywell Hill, St Albans, Herts AL1 1EX.
Telephone: St Albans (0727) 40303
(24hrs answerphone service for payment by Connect, Access or Visa card)